图灵程序设计丛书

WILEY

[美] Eric A. Meyer 著
姬光 译

Smashing CSS

Professional Techniques for Modern Layout

精彩绝伦的CSS

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

[美] Eric A. Meyer 著
姬光 译

Smashing CSS

Professional Techniques for Modern Layout

精彩绝伦的CSS

人民邮电出版社
北京

图书在版编目（CIP）数据

精彩绝伦的CSS / （美）迈耶（Meyer,E.A.）著；姬光译. -- 北京：人民邮电出版社，2012.6
（图灵程序设计丛书）
书名原文：Smashing CSS:Professional Techniques for Modern Layout
ISBN 978-7-115-28479-2

Ⅰ. ①精… Ⅱ. ①迈… ②姬… Ⅲ. ①网页制作工具 Ⅳ. ①TP393.092

中国版本图书馆CIP数据核字(2012)第120414号

内 容 提 要

本书是国际知名CSS大师写就的实用教程，分为3部分。第一部分介绍了CSS基础知识，对一些常用工具和基本技术进行了介绍，包括CSS选择器等。第二部分展示了各种CSS效果，包括同一效果的不同实现方式，并讲解了布局设计。第三部分介绍前沿技术，主要是最新HTML5和CSS3技术。

本书适合所有Web开发与设计人员学习参考。

图灵程序设计丛书

精彩绝伦的CSS

◆ 著　[美] Eric A. Meyer
译　姬　光
责任编辑　毛倩倩

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　印刷

◆ 开本：800×1000　1/16
印张：14.75　彩插6
字数：349千字　2012年6月第1版
印数：1－4 000册　2012年6月北京第1次印刷

著作权合同登记号　图字：01-2011-1847号

ISBN 978-7-115-28479-2

定价：49.00元

读者服务热线：(010)51095186转604　印装质量热线：(010)67129223

反盗版热线：(010)67171154

版权声明

献给Kathryn、Carolyn和Rebecca。

前　　言

CSS已经发展得非常成功了，甚至像HTML一样成功！尽管有时候它的确令人难以掌控。现在无论是在浏览器还是应用商店，甚至聊天客户端，CSS都无处不在并且没有任何消退的迹象。随着这门语言的应用愈加广泛，其能力也在不断增强。

本书涵盖了近100个使用CSS制作优秀网站的窍门、技术、工具以及一些技巧，其中每个部分都是相对独立的。你可以随便翻开一页就读，而不用担心会错过前面章节中的某些重要内容。本书假定你至少已经熟悉一点儿CSS和CSS的使用，这种假定的熟练程度可以描述为“进阶的初学者到中级的使用者”。所以，如果你一点儿也不了解CSS，或者知道的比那些撰写规范的人都多，那本书恐怕不适合你。如果你不是这两类人，本书还是有很多值得学习和享受的内容的。

在本书的第一部分，我们对一些常用的工具和基本技术进行了介绍，其中还包括一些不那么好理解的CSS选择器。第二部分展示了CSS可以实现的各种各样的效果，其中包括一些好玩的特效、对同一目标的不同实现方式以及布局等。第三部分介绍的是前沿技术，这些技术或许你现在的项目里还用不到，但是随着时间的推移，它们一定会在你的工作中变得越来越重要。

请访问本书的网站www.wiley.com/go/smashingcss下载代码示例[①]。

大约十年以前，你可能还以为CSS大限将至了。但是到了2010年，它却变得更加充满活力且比以往更加引人注目了。我希望你能享受这前后封皮之间的内容，就如同我写作此书时同样地享受。

① 也可在图灵社区（www.ituring.com.cn）本书网页免费注册下载。——编者注

致　　谢

感谢Chris Webb拉我“入伙”并且耐心地忍受着我每一次的拖延和耽搁。有那么几次，项目看起来都要停掉了，Chris却总能以他良好的幽默感和无限的淡定督促项目前进。我非常尊敬他，而且我们还计划把某一年的暑假安排在一起，这样我们就可以坐在游泳池边一起举杯喝鸡尾酒了。

感谢Debbye Butler和Brian Herrmann在编辑、审稿期间对书稿的润色提升，并且详细指出了行文中的瑕疵，以及我含糊的解释。

感谢我忠实的读者们，无论是实体书的读者还是网络上的读者，我都真心地感谢你们，非常感谢。

感谢我的妻子和女儿，对你们的谢意我无以言表。

Eric A. Meyer

2010年8月13日，俄亥俄州克利夫兰高地

目　　录

第三部分 前沿技术

Part 1

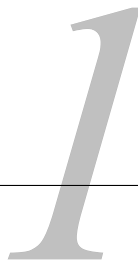
第一部分

基本原理

本部分内容

第 1 章

工　　具



无论做什么事情，工具都能起到很大的辅助作用，创建网页或者应用亦如此。对于CSS来说，既有可以帮助我们书写CSS的工具，也有使用CSS构建的工具来辅助我们进行开发。甚至有的工具可以使我们的浏览器支持更多原生并不支持的CSS特性，你不仅是开发者，还是个能工巧匠！相信本章介绍的这些工具绝对会撑爆你的工具箱。

1.1　Firebug

Firebug（见图1-1）是任何网页开发人员的工具箱中不可或缺的两个工具之一（关于另外一个，请参考1.2节）。它是火狐浏览器（Firefox）中一个完全免费的扩展，如果用的是其他浏览器，也可以接着往下看，因为你一样可以使用Firebug！

图1-1　Firebug主页

要想安装Firebug，可以在火狐浏览器中访问getfirebug.com，然后单击Install（安装）按钮（写作本书时，这个按钮就在网页的右上方）开始安装，安装完成后重启火狐浏览器即可。现在准备开始你的神奇之旅吧！

我没法在这么短的篇幅中讲解Firebug的全部功能。实际上，即使整整一章的篇幅都未必够用，我这里只讲一些重点。

图1-2中所示的HTML选项卡左侧展示的是文档结构，单击箭头可以展开或收缩文档的子结构。注意在该选项卡中，当把鼠标悬停在某个元素名上时，该元素会在页面中突出显示。最神奇的是，它还可以显示彩色区域并利用不同色彩展示元素的内边距（padding）和外边距（margin）。例如，本例中的内容区域为浅蓝色，内边距是淡紫色，外边距是浅黄色。具体是什么颜色其实不重要，关键是在页面上可以很直观地看到效果。

图1-2 通过Firebug查看元素的布局（另见彩插图1-2）

在HTML选项卡的右侧，可以通过单击Style（样式）选项卡查看应用在当前元素上的CSS（见图1-3）。这里不仅包含你自己写的样式，还包含浏览器自身的内建样式。例如，你可以看一下html.css和quirk.css这两个文件的内容，这些就是内建样式（这些样式称为“用户代理样式”，可以通过点击Style选项卡，在弹出的菜单中选择是否显示用户代理样式）。

有一点需要注意，Firebug有时候会显示一些像-moz-background-clip这种未曾声明过的属性，如果确定没有明确声明那些属性，基本就可以将其忽略掉。另外，如果你使用的是简写形式的属性，它也会自动扩展成独立的属性，也就是说像这样的代码：

```
font: 1em "Andale Mono", "Courier New", Courier, monospace;
```

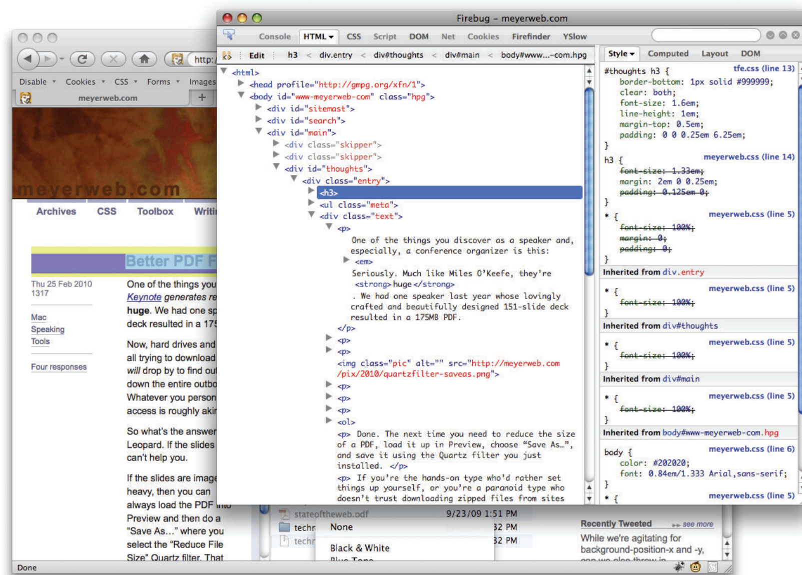


图1-3　在独立的Firebug窗口中Style选项卡全部展开

在Firebug中就会变成这样：

```
font-family: "Andale Mono","Courier New",Courier, monospace;
font-size: 1em;
font-size-adjust: none;
font-stretch: normal;
font-style: normal;
font-variant: normal;
font-weight: normal;
line-height: normal;
```

尽管刚开始可能有点儿令人困惑，但是这种呈现方式并不赖，因为它可以提醒你在这些简写的背后还隐藏着一些东西（更多的简写属性见第2章。）

另外需要注意的是，Style选项卡中列出的规则是按特殊性（specificity）降序排列的，也就是说，第一个是应用在当前元素上最特殊的规则（优先级最高），第二个是特殊性稍低一点的规则，依次类推。（更多关于特殊性的内容，详见2.3节。）

通过随时单击鼠标右键并在弹出的菜单中选择Inspect Element（审查元素）选项可以查看任何元素（如图1-4所示），你也可以单击Firebug图标旁边带箭头的方框图标切换到审查模式。此时，随着鼠标在页面上的移动，当前元素会显现出轮廓，单击元素即可进行查看。

在Firebug中单击任何声明左侧的空白处都可以禁用该声明，这种禁用声明的方式对于测试各个属性之间的相互作用非常有用。在图1-5中还可以看到，鼠标悬停在颜色值上会出现一个小色块，它用来展示该值对应的颜色。

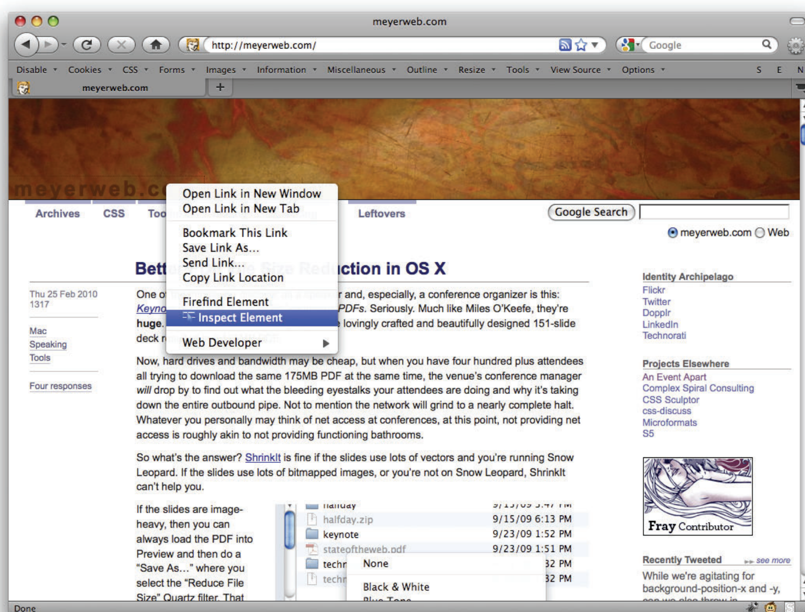


图1-4 右键菜单中的Inspect Element选项

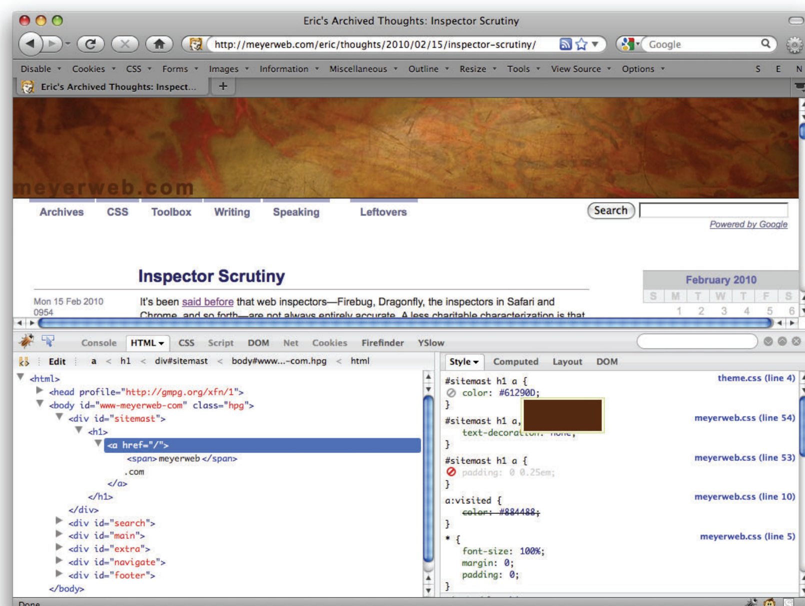


图1-5 禁用的样式和悬浮的颜色框（另见彩插图1-5）

如图1-6所示，在Firebug的Style选项卡中也可以查看元素的计算样式。这就意味着，无论你是否曾经声明过，它都会把所有已知的应用在该元素上的CSS属性展示出来。记住，所有的CSS属性都有默认值。在该视图下可以查看全部的默认值，例如当你想知道浏览器应用在标题上的行高（`line-height`）的确切像素值时，该视图就变得非常有用了。

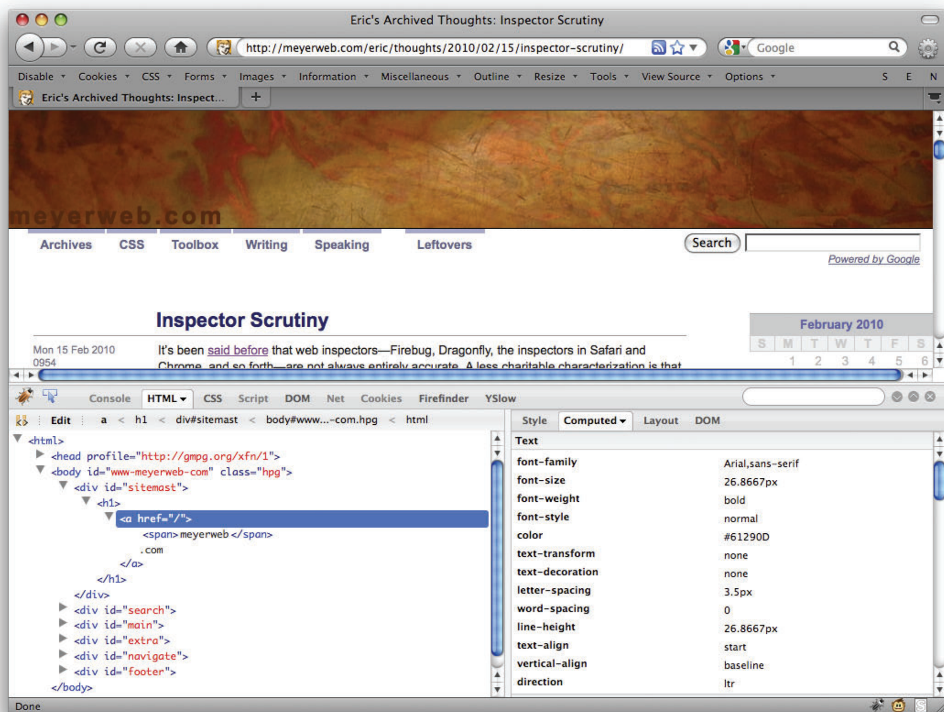


图1-6　计算样式

通过查看元素的盒模型（box model）部分，可以精确地查看元素的尺寸大小，如元素的宽、高、内边距和外边距等（如图1-7所示），这些都是用像素表示的。更酷的是当鼠标悬停在该面板中的框上时，页面上就会出现沿着元素外框的上边缘和左边缘放置的像素尺。

在图1-8中我们可以很明显地看到，Firebug还有许多其他功能，诸如可以编辑元素的属性值（比如`class`）或者元素本身的内容、添加或编辑CSS属性和值等。通过在Firebug界面中随时单击鼠标左键或者右键可以自行探索Firebug的功能，看看你还能做点儿什么？

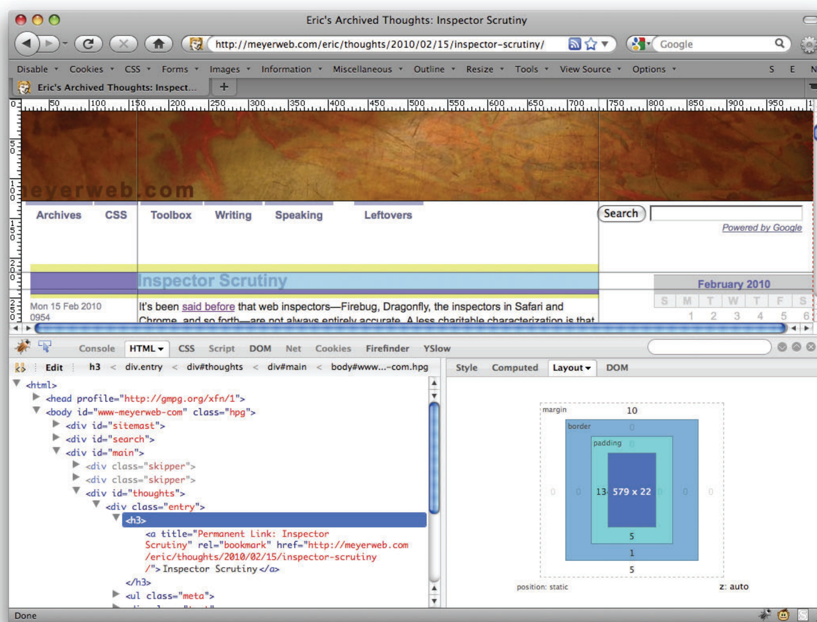


图1-7　Layout（布局）选项卡（另见彩插图1-7）

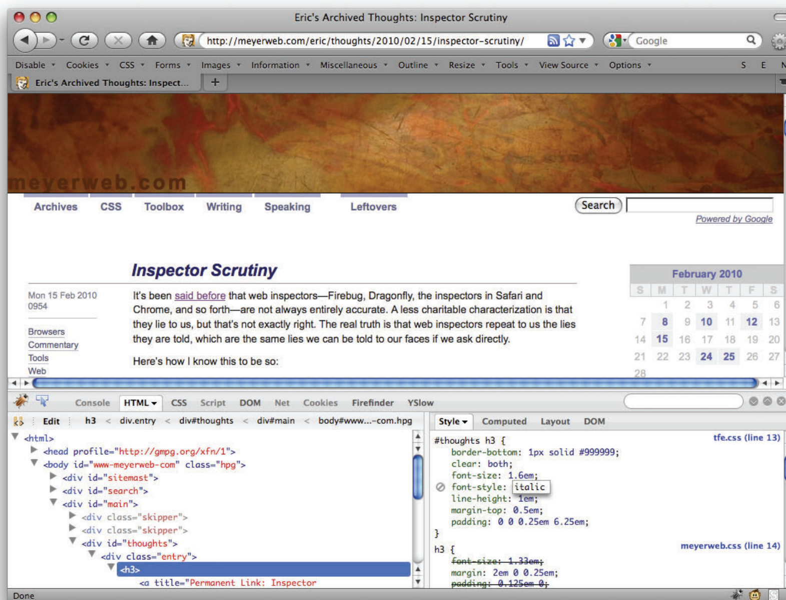


图1-8　在Style选项卡下动态编辑CSS

需要注意，当在Style选项卡下查看元素的CSS时，无法看到任何伪元素（pseudo-element）影响元素的相关规则。作为例子，如果我们使用了选择器`p:first-letter`，那么当查看p元素时就无法看到这条规则，伪类（pseudo-class）是可以看到的，但是伪元素不能。这对于包含生成内容的清除浮动（clearfix）解决方案，可能是个问题（见4.5节）。

如果你没有使用Firefox进行开发，但却想一睹Firebug的风采，那么访问页面getfirebug.com/lite.html（如图1-9所示）并且按照页面上的指引进行安装，就可以使Firebug运行在Internet Explorer、Opera或者Safari等浏览器中，以此来适应你的开发环境。你可以把Firebug精简版链接到测试页面进行测试，或者把它保存到书签栏存成一个书签小程序（bookmarklet）——这也是我推荐的方式。

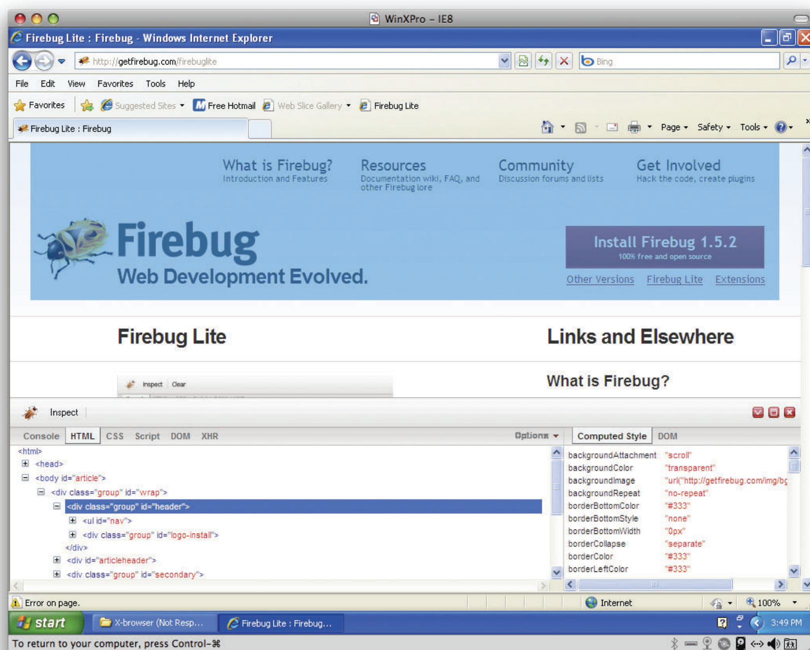


图1-9　Firebug精简版在IE浏览器上运行

这个版本的Firebug并不像火狐扩展版的功能那么多，因此才有“精简版”一说，但是它仍然很好很强大。

1.2　Web Developer Toolbar

除了Firebug，另外一个网页开发人员必不可少的工具就是WDT（Web Developer Toolbar）了，它也是火狐浏览器的一个完全免费的扩展。

访问chrispederick.com/work/web-developer可以获得此扩展。你也可以访问addons.mozilla.org，然后搜索“Web Developer Toolbar”，在WDT的安装页面进行安装（如图1-10所示）。

图1-10　addons.mozilla.org网站上的Web Developer Toolbar页面

跟Firebug一样，我也没法把WDT的所有功能都讲一遍，那样的话一整章也不够用。这里只挑选了其中几个菜单进行重点讲解，当然，当你装完WDT之后应该花点儿时间看看所有的菜单和选项。

有时候需要频繁更新一些小的改动，而浏览器缓存可能变得很烦人，这时候禁用浏览器缓存就很有用了。如图1-11所示，还可以通过WDT禁用JavaScript，它可以让你看到当全部脚本都停止工作或者JavaScript框架没有载入时会发生什么。

如图1-12所示，CSS菜单中的某些功能在Firebug中也有，但是WDT有个非常棒的功能——可以只禁用嵌入样式（embedded style）、链接样式表（linked style）或者只禁用行内样式（inline style），当然你不应该使用行内样式（inline style）！如果你想看到一个畸形的页面，甚至可以用WDT关掉大部分的浏览器内建样式。

图1-13中的Information（网页信息）菜单中包含了大量好玩的小报告，包括显示文档中的类（class）和ID信息、显示`div`的顺序、网页中的颜色信息汇总，等等。你还可以启用显示元素信息模式，当单击任何元素时都可以看到该元素各属性和值的汇总信息，诸如元素在页面中的位置、字体、元素的祖先和后代元素，等等。Information菜单展示的内容跟XRAY很像，本章后面会介绍更多关于XRAY的内容。

图1-11　Disable（禁用）菜单

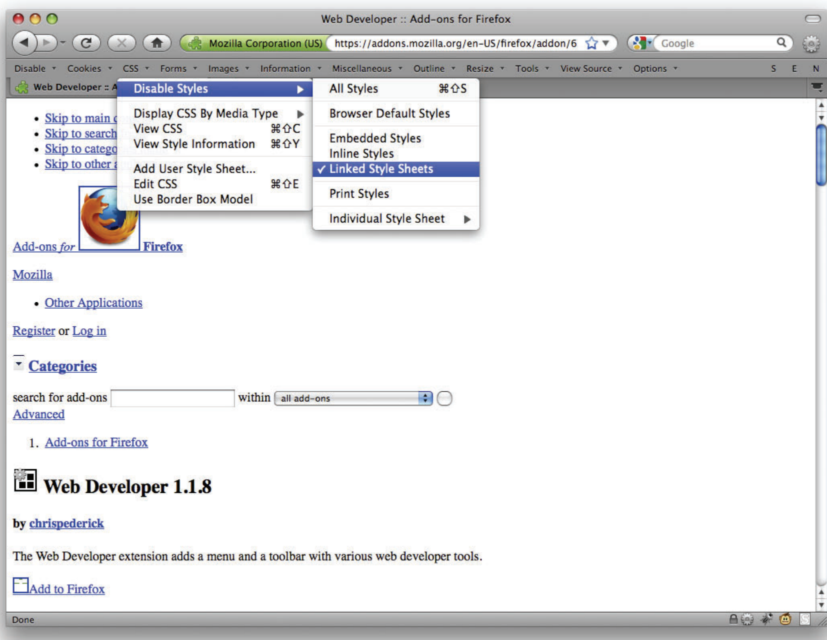


图1-12　通过CSS菜单禁用链接样式表

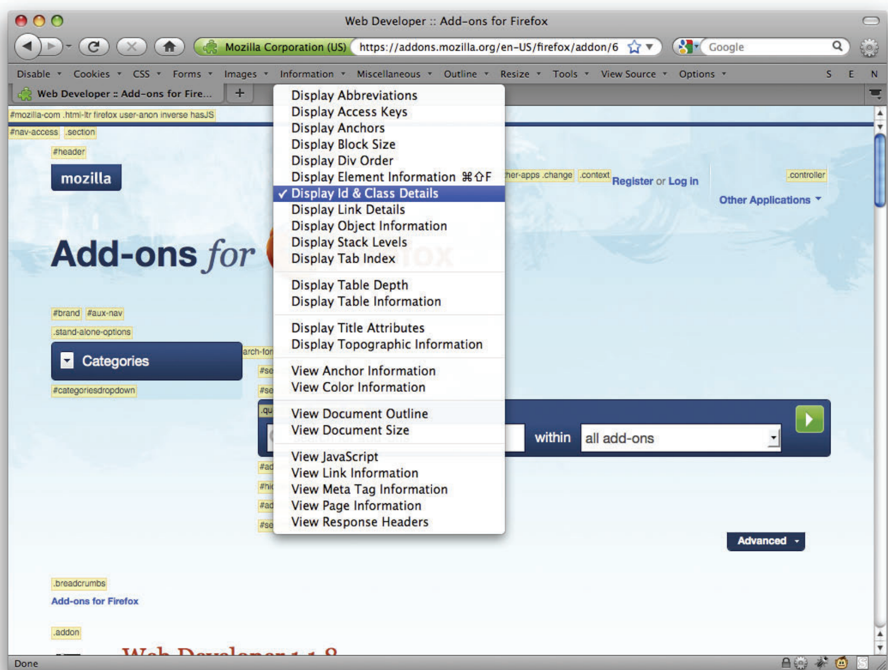


图1-13　通过Information菜单显示类和ID的值

通过图1-14中所示的Outline（标示[①]）菜单可以标示出各种元素，包括全部块（block）元素、全部行内（inline）元素、全部链接、全部定位元素、全部表格单元格等，还可以自定义需要标示的元素以及标示颜色。当标示元素时，你也可以选择是否在页面上显示元素的名字。这个菜单可比它看上去要强大得多，我把它当做一个Layout Diagnostic（布局诊断）菜单来用，因为我可以快速标示出不同的元素集合并且快速查看它们之间的叠放关系，快速找出可能出错的地方。

Tools（工具）菜单提供了一些验证器、错误检查器和调试控制台的快捷访问方式，而其中最好的一个特性就是可以验证本机的HTML和本机的CSS（如图1-15所示，分别对应“Validate Local HTML”和“Validate Local CSS”项），在这两种情况下，当前浏览的网页都会被打包成序列化的字符串发送到相关的验证器。因此，如果选择了Validate Local HTML，页面的标记就会被发送到HTML验证器，然后你会看到一个报告。这对于验证那些受防火墙保护的页面和公网无法访问（因此验证器服务也无法访问）的页面来说非常实用，有了本机验证功能，这些都不再是问题了。

① 此处翻译参照了Web Developer Toolbar中文界面的菜单项。（本书脚注若无特殊说明均为译者注。）

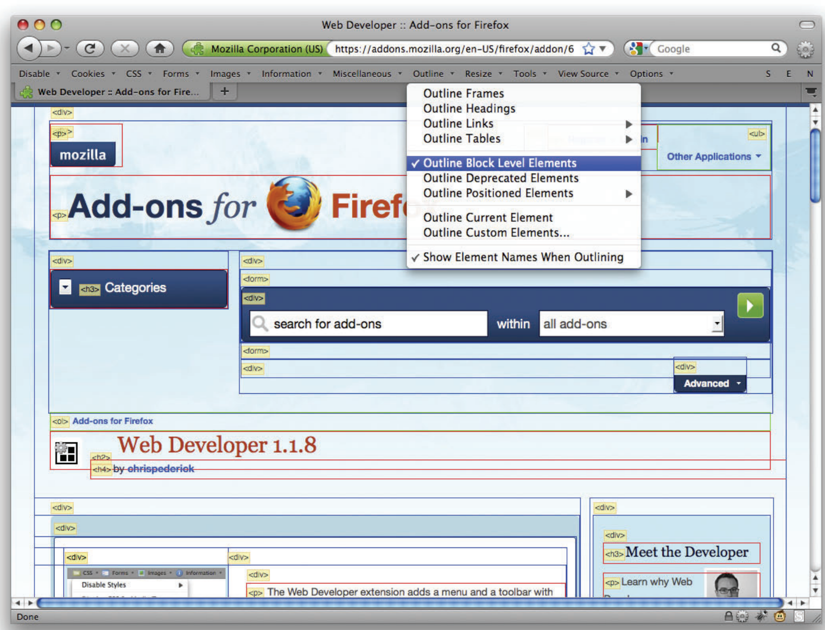


图1-14 使用Outline菜单标示出块元素

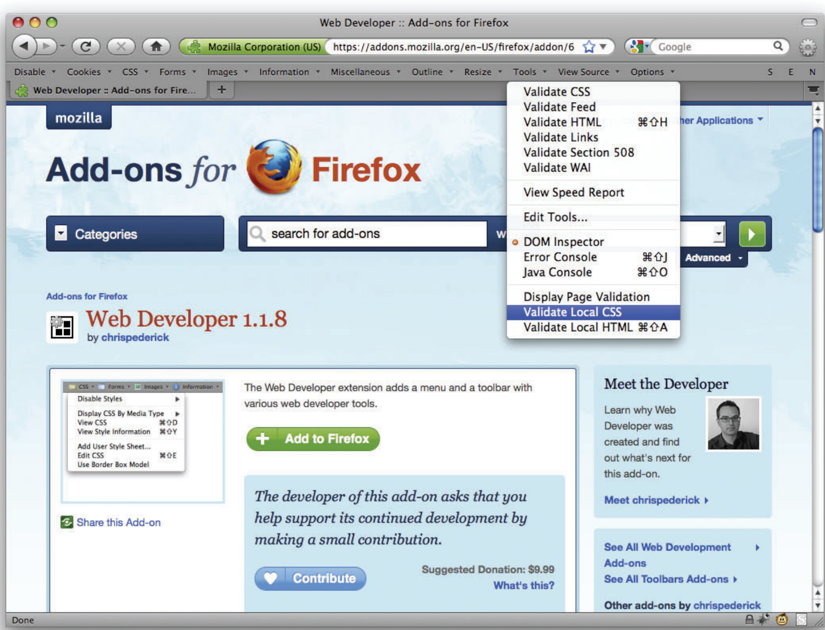


图1-15 Tools菜单

正如我开始所说的，对于WDT的全部功能来说这些只是管中窥豹，所以花点儿时间仔细研究一下这个工具吧，它会让你的生活变得更轻松！

1.3 IE 开发者工具栏

如果你主要用Internet Explorer 7完成开发工作，那就不能安装Web Developer Toolbar了，但你可以安装IEDT（Internet Explorer Developer Toolbar，IE开发者工具栏）来作为替代品。

IEDT的URL地址是那种非常典型的谁也记不住的微软URL，所以打开你最喜欢的搜索引擎（使用Google会有那么一点儿讽刺），然后输入“Internet Explorer Developer Toolbar”，应该第一个搜索结果就是了，如果是IE7的话可以继续单击进行安装。IEDT不支持IE8，一会儿我们再看看IE8都提供了什么。

完成安装后，你就可以找到右上角的Tools菜单（如图1-16所示，就在Pages（页面）菜单附近），注意不是左侧Favorites（收藏）和Help（帮助）菜单之间的那个Tools菜单。在该菜单下选择Toolbars（工具栏），然后选择Explorer Bar（浏览器栏），最后再选择“IE Developer Toolbar”即可。

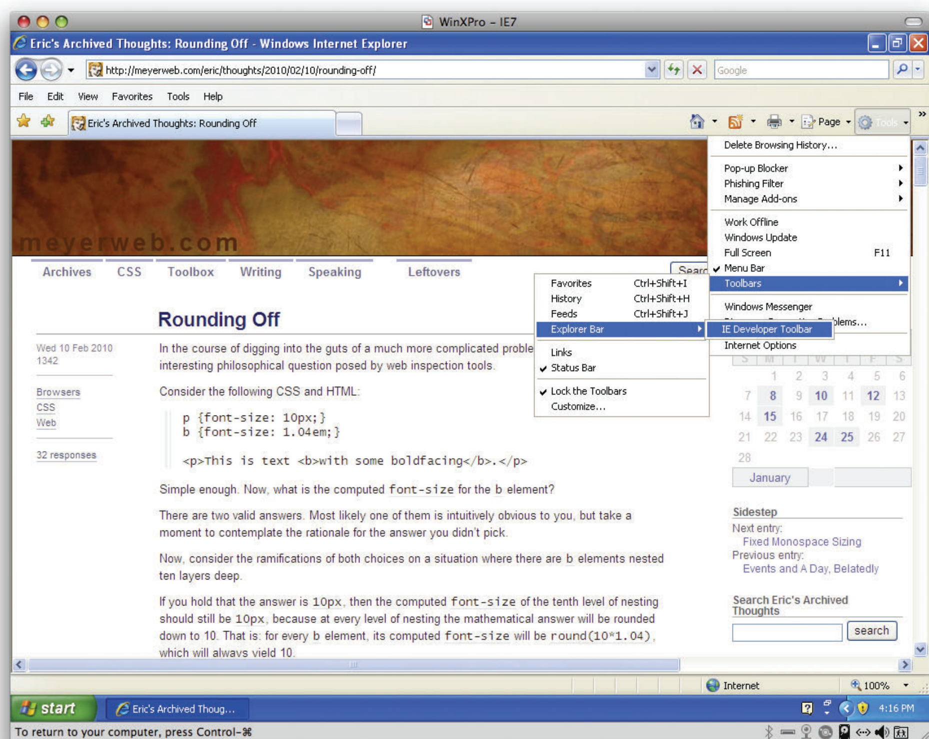


图1-16 IE开发者工具栏在IE7右上角

单击IE Developer Toolbar后，你就会看到在浏览器窗口底部打开了一个像Firebug一样的面板，如图1-17所示，在面板顶部也有一些跟Web Developer Toolbar很像的菜单。单击面板右上角关闭按钮附近的层叠窗口小图标，可以使面板扩展为一个独立的窗口，通过这种方式扩展面板对于一些低分辨率的上网本和高射投影仪来说特别有用。

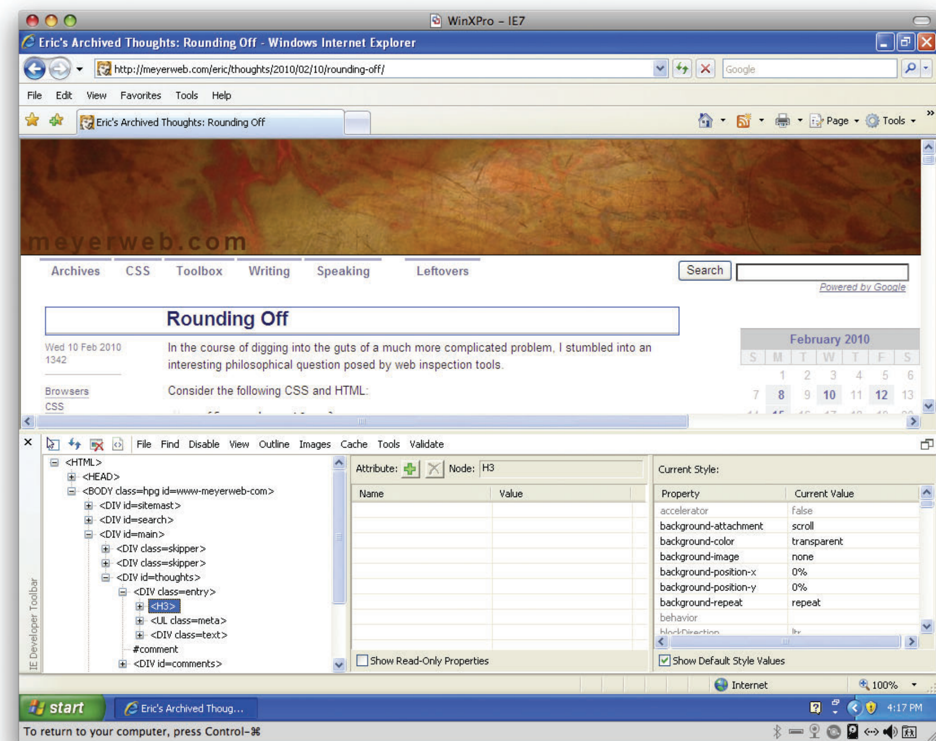


图1-17　运行中的IE开发者工具栏

这个工具栏有个很好的功能，就是通过Show Default Style Values（显示默认样式值）复选框可以很容易地在计算样式和声明样式之间进行切换（如图1-18所示）。同样，通过Show Read-only Properties（显示只读属性）复选框可以查看当前元素的DOM（Document Object Model，文档对象模型）属性的每个细枝末节。如果你不熟悉JavaScript和DOM脚本编程，这些可能还不适合你（当然，它也不适合我）。

IE开发者工具栏包含了Web Developer Toolbar的部分功能，但把大部分最有用的功能放在了第一层菜单中，诸如标示元素和验证本机的HTML和CSS。查看菜单中还有一个很小巧的功能，即CSS选择器匹配（如图1-19所示），它会弹出一个窗口展示CSS中的每条规则及其匹配了文档中的多少元素，如果有的规则显示0匹配，则说明该规则没有匹配页面上的任何元素，可以考虑删掉这条规则。

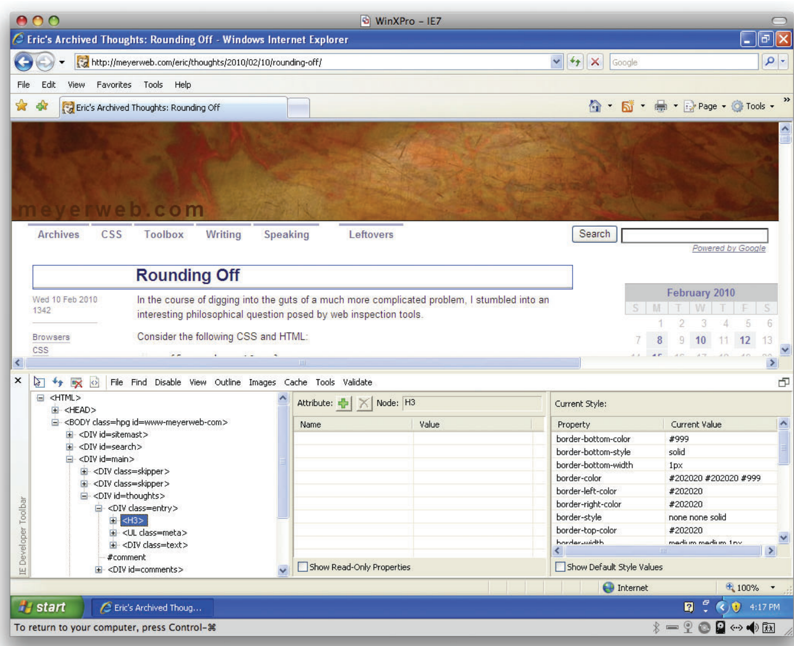


图1-18　未勾选Show Default Style Values时的IE开发者工具栏

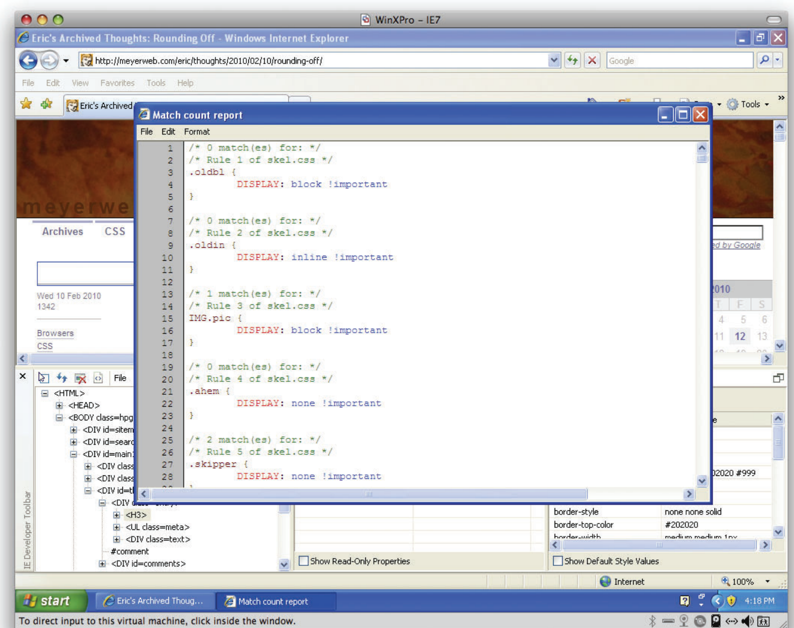


图1-19　选择器匹配报告

IE8内置了一个开发人员工具，所以就不用额外安装了。帮助文档是在线的，截至撰写本书之时，该文档仍放在一个比IEDT的URL强不了多少的URL地址上。所以还是打开你最喜爱的搜索引擎，然后输入“Discovering Internet Explorer Developer Tools”，第一个搜索结果应该就是该文档了。

想打开这个工具的话，在IE的Tools菜单中选择Developer Tools（开发人员工具，如图1-20所示）或者按键盘上的F12健即可。可以看到它跟IE Developer Toolbar的界面很像，可以说是Firebug和Web Developer Toolbar混合的产物。它的菜单大部分和IE7的开发者工具栏很像，但是跟IE7的开发者工具栏相比，那些选项卡却更像Firebug。

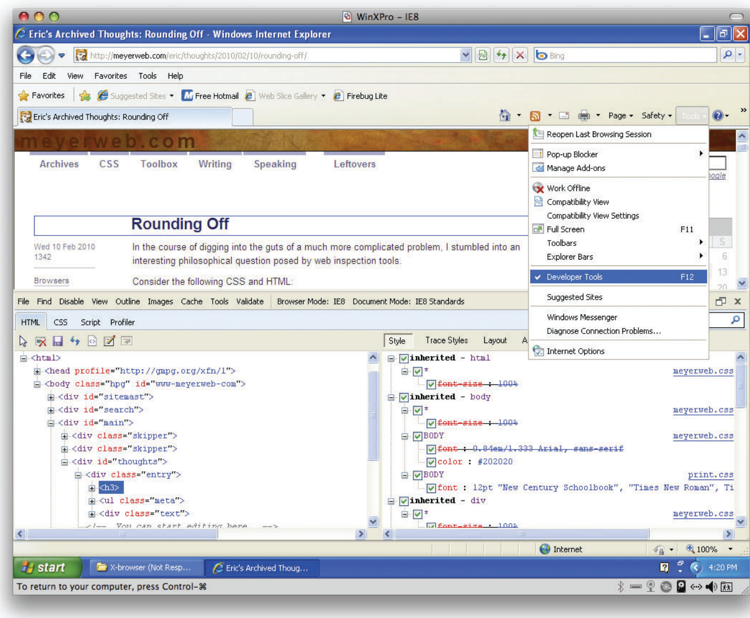


图1-20　IE8的开发人员工具

对于开发人员工具的Style（样式）选项卡来说有一件事让我感到很困惑，就是样式的排列顺序对我来说没有任何意义，它不是按照特殊性顺序排列的。尽管这个列表确实显示了哪些声明被其他声明覆盖了（这也挺好的），但是不像Firebug那样有明确的顺序，不太好用。

尽管如此，即便开发者工具栏和开发人员工具不能涵盖Firebug或Web Developer Toolbar的全部功能，它也还是很有用的。任何网页开发人员都应该把它装在Internet Explorer上，它对于查看布局的源代码和追踪IE的各种怪异错误非常好用。

1.4　Dragonfly（Opera 浏览器）

如果主要用Opera浏览器进行开发，那么你会用到Dragonfly（如图1-21所示），它是一个Opera 9.5及以后版本内置的开发环境，访问opera.com/dragonfly可以获得更多信息。

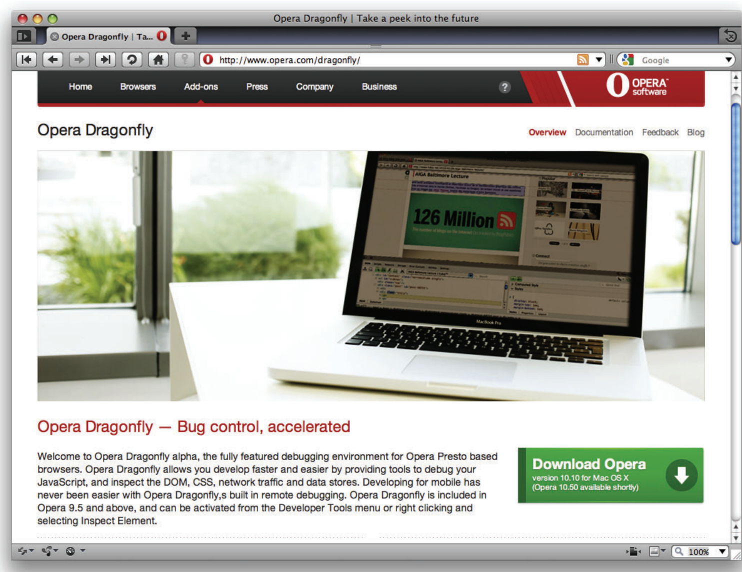


图1-21　Dragonfly的主页

打开Dragonfly的默认方式是单击Tools菜单，然后在Advanced（高级）选项下选择Developer Tools（开发者工具），你还可以安装一个Debug（调试）菜单，访问opera.com/dragonfly可以找到安装链接[①]，一旦安装完成，就有了一些访问Dragonfly和其他功能的快捷方式。当然，你也可以使用键盘快捷键打开Dragonfly，Mac快捷键是Option+Command+I，Windows是Option+Control+I[②]。但奇怪的是，这里的键盘快捷键是不能进行切换的。如果已经打开了Dragonfly，就不能使用键盘快捷键关掉它了。这种情况下就需要使用鼠标或者Command+W（Control+W）关掉它，这在Dragonfly作为独立的窗口打开时很管用，但是当Dragonfly停靠在浏览器窗口时则仅当单击Dragonfly使其获取焦点时Command+W才能关掉它，否则会关掉整个窗口[③]。

Debuy菜单有个很好的特性，它包含了到HTML、CSS和其他规范的链接。还有个好玩的地方是它的Layout（布局）子菜单，可以将Opera设置成像Emulate Text Browser（模拟文本浏览器）和Show Structural Elements（显示结构元素）那样的布局模式，甚至还有一种Nostalgia（怀旧）布局模式（如图1-22所示），它可以温暖任何一个80年代计算机老兵的心。

虽然Dragonfly的界面布局跟Firebug的非常相似，但也有一些显著的不同。首先，右侧的Style选项卡可以显示计算样式和声明的样式（如图1-23所示），每个分组都可以扩展和收缩。跟Firebug一样，你看到的也不完全是声明的属性，因为简写的属性也同样被扩展成了独立的属性。Dragonfly有个很好的体验，就是如果你想查看简写属性，那么至少在计算样式里可以看到。

① 在翻译本书时，最新的主页上已经找不到安装链接了。
② Windows下Opera最新版本11.60的快捷键应为Ctrl+Shift+I。
③ 最新版本Opera 11.60已经修复该问题，可以使用快捷键进行切换。

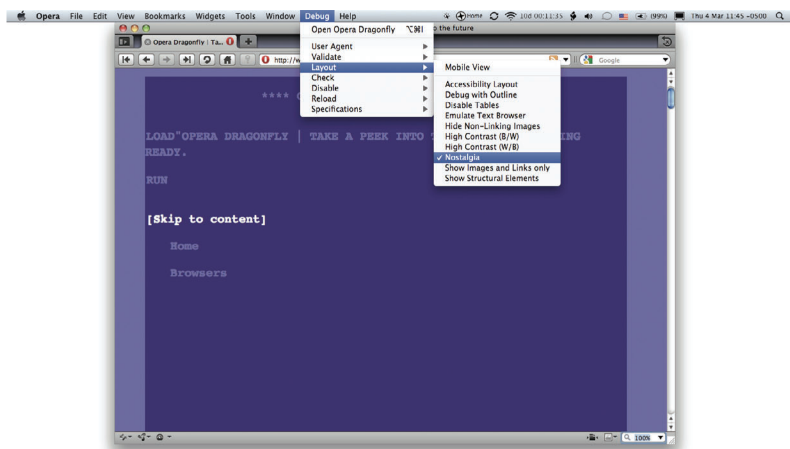


图1-22　Nostalgia视图下的Dragonfly页

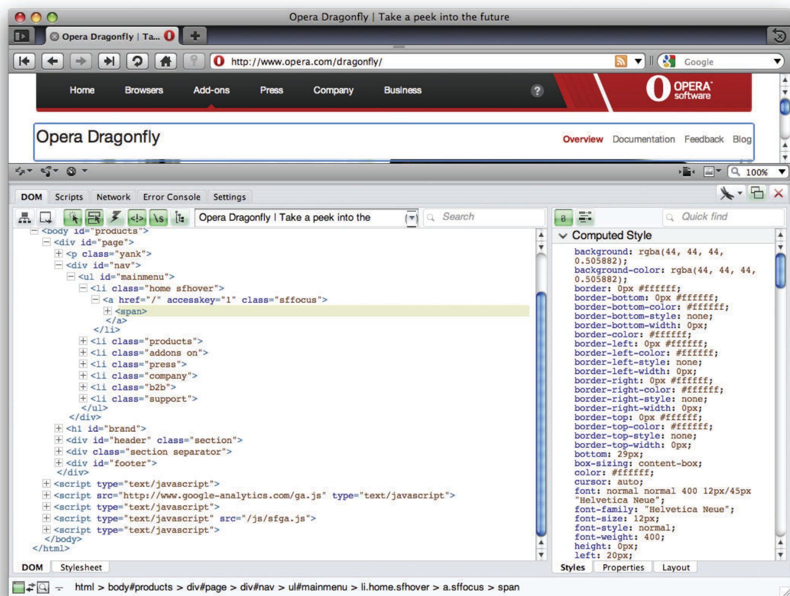


图1-23　计算样式分组展开时的Dragonfly

另外一个与Firebug的差异（不太受欢迎）是，任何被其他声明覆盖的声明都会灰掉，而且有个橘黄色的[overwritten]文字跟在后面（如图1-24所示），这会显得很乱，而且很难看清被覆盖的声明的值[1]。

① 最新版Opera 11.60中的Dragonfly已经解决该问题，此处体验与Firebug一致。

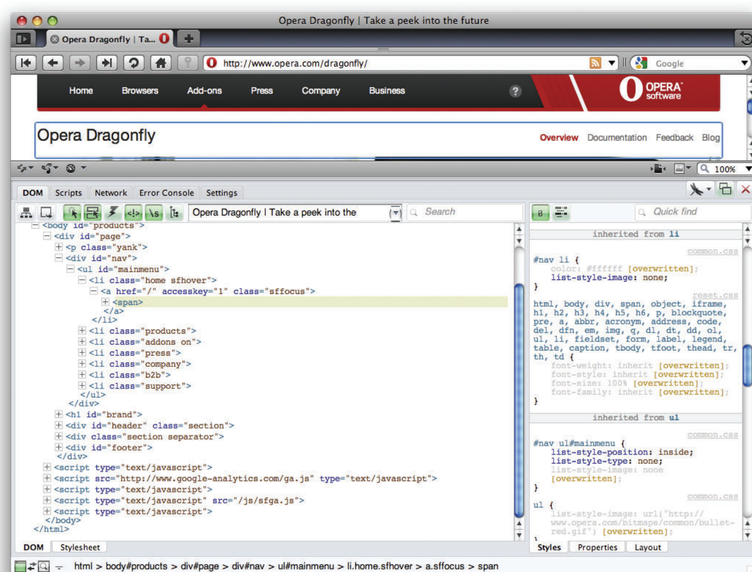


图1-24 Styles选项卡中展示被覆盖的样式（另见彩插图1-24）

图1-25展示了Layout（布局）选项卡，它可以显示当前被检查元素的布局框，这里不仅可以显示布局框的大小，还能显示许多属性的像素值，比如`offsetTop`和`scrollLeft`等。

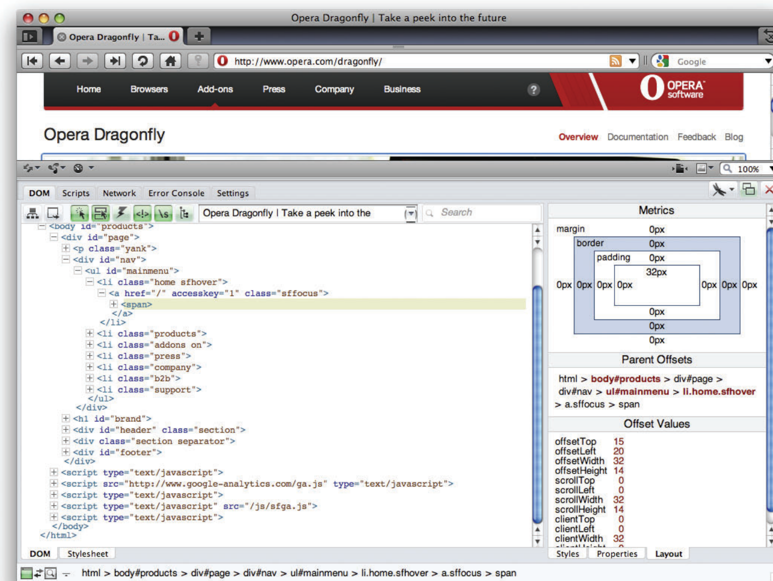


图1-25 Dragonfly详尽的Layout选项卡

1.5　Web 检查器（Safari 浏览器）

如果你主要用Safari浏览器进行开发，那么就会用到Web检查器（Web Inspector）。

要想使用Web检查器，打开Safari的Preferences（偏好设置）选择Advanced（高级），然后将Show Develop menu in menu bar（在菜单栏中显示开发菜单）前面的复选框勾选上，如图1-26所示。当完成这些之后，你就可以通过单击Develop（开发）菜单下的Show Web Inspector（显示Web检查器）打开Web检查器了，或者通过键盘快捷键Option+Command+I开启[①]。跟Dragonfly一样，Web检查器的键盘快捷键也不能进行切换，如果Web检查器已经打开的话就不能用快捷键关闭它了。这种情况下需要使用鼠标，而Command+W只有在Web检查器作为独立窗口打开时才起作用，如果Web检查器停靠在主窗口，那么使用Command+W的话就会关掉整个窗口。

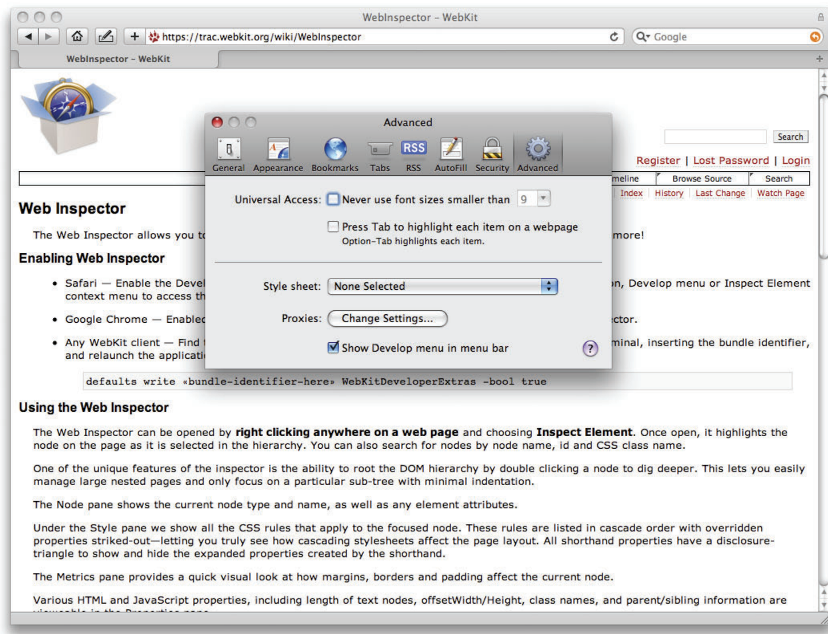


图1-26　开启Develop菜单

尽管Web检查器的界面布局跟Firebug的非常像，但也有一些显著的不同。比如右侧的面板将Computed Style（已计算样式[②]）作为一个分组，如图1-27所示。跟Firebug一样，这里看到的也不全是声明过的属性，因为简写的属性也会被扩展成独立的属性，并且如果选择了Show Inherited（显示继承的样式）就会得到一个很长的属性列表。

在它的正下方，当前被检查元素的每条规则都以独立的分组显示。可以分别展开或收起每个

① Windows下的Safari快捷键为Ctrl+Alt+I。

② 面板上的各个中文标签参照Safari中文版界面。

分组，而这些分组的下方是Metrics（版式）子面板，它展示了当前被检查元素的布局框的尺寸（如图1-28所示）。

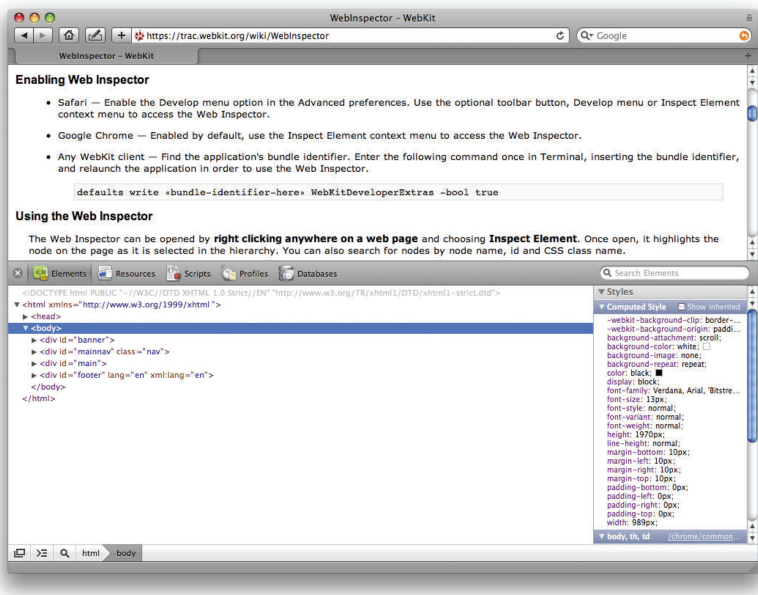


图1-27　已计算样式

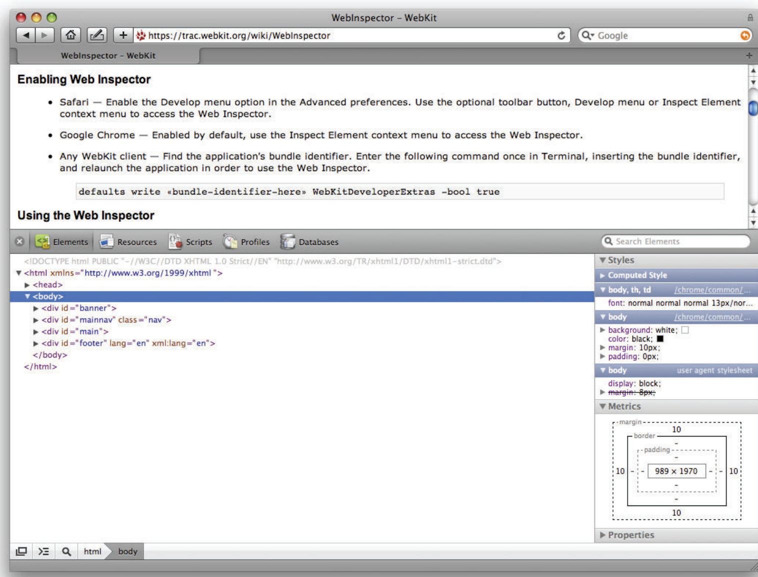


图1-28　常态下的样式和布局分组

1.6　XRAY

如果你正想找一个轻量级的、跨浏览器的元素检查器，那么图1-29中所示的XRAY一定“很合你的胃口”。它的功能非常有限，但如果你不需要过多的功能，那么对于功能的专注其实正是它的强项。

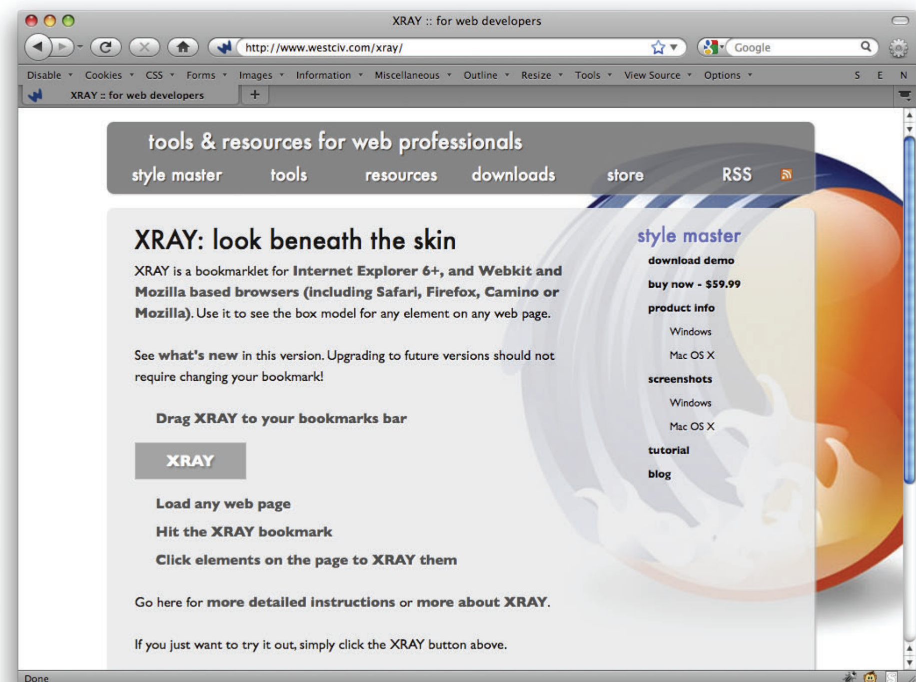


图1-29　XRAY的页面

打开westciv.com/xray，直接把那个大大的XRAY按钮拖放到你的书签栏（或者拖到菜单栏，如果你想把它隐藏的话）。然后，任何时候当你打开一个页面并且想检查某个元素时，打开XRAY，然后选择感兴趣的元素即可。

选择元素之后，元素会突出显示且其各边就会出现尺寸信息，你还会看到XRAY框，如图1-30所示。XRAY框里面会提供一些额外的信息，诸如元素在文档树中的位置、ID或类的值、核心的CSS属性值等。如果选择了继承结构（inheritance hierarchy）的任何元素，XRAY会继续检查该元素[1]。单击XRAY框右上角的关闭图标即可退出XRAY，直到下次需要的时候再打开。

还有一个类似但用途不同的工具，即MRI（westciv.com/mri），它允许你输入选择器，然后就可以在页面上显示都有哪些元素被选中了。

① XRAY会一直检查你当前单击的最内层的元素，即单击事件发生的元素。

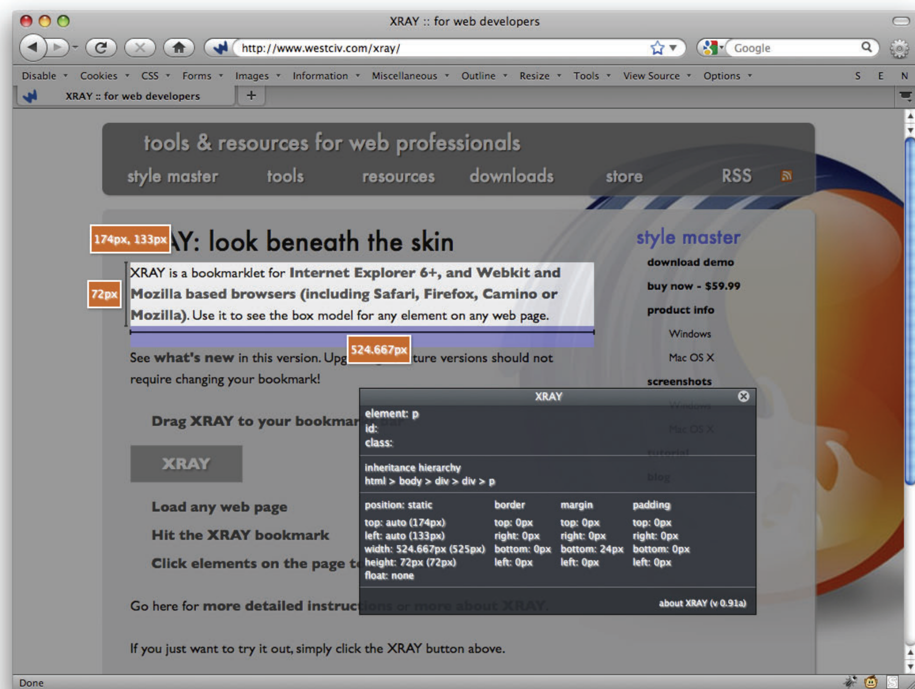


图1-30　运行中的XRAY

1.7　SelectORacle

如图1-31所示，SelectORacle这个名字听起来像是某个数据库产品的广告，但它真不是！相反，它是一个可以将有效的选择器转换成类似常规英语的在线工具。这个名字源自于“选择器”（selector）和“预言”（oracle）的组合。

打开gallery.theopalgroup.com/selectoracle并且输入一个或多个有效的CSS选择器，随便多复杂都行。选择英语或者切换到西班牙语，然后单击Explain This!（解释）按钮就会得到每个选择器的完整解释。例如：

```
ul li:nth-child(2n+3):not(:last-child)
```

会被解释成：

选择ul元素的后代元素中，从第三个子元素开始的索引为奇数的全部li元素，且不包含最后一个li元素[①]。

① 该工具主要面向英文用户，目前还没有中文版。

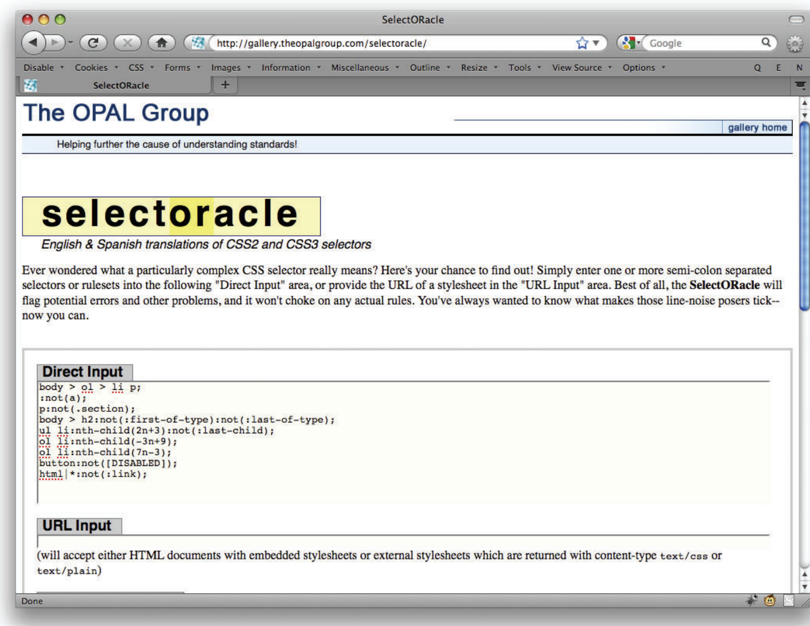


图1-31　SelectORacle页面（另见彩插图1-31）

OK，或许这乍一看有点儿令人困惑，但是如果你慢慢读的话，大多数情况下都能读明白。当然，截至撰写本书之时，世界上还没有任何一种浏览器能够完整支持这一选择器，所以不必太在意。你只需要知道，当遇到一个可能表意不清的选择器时，SelectORacle或许能帮上忙。

你曾想过当浏览器载入了过时的页面，或者某些元素出了问题（比如font元素）时，浏览器可以喊你一下吗？现在诊断样式表（diagnostic style sheet）就可以做到，当然，只是视觉上的提醒。

1.8　诊断样式表

当页面上的标记出了问题时，诊断样式表可以快速给出视觉提示。在meyerweb.com/eric/tools/css/diagnostic可以找到一个使用诊断样式表的例子（有CSS3和IE7友好的版本），还有个类似的资源在accessites.org/site/2006/07/ big-red-angry-text。

那么它有什么用呢？例如，meyerweb（meyerweb.com）的诊断样式表中有这么一行：

```
*[style], font, center {outline: 5px solid red;}
```

它会把任何含有style属性的元素及任何字体（font）元素和居中（center）元素用一条红色的粗实线圈起来。你还可以给它再加点儿“佐料”，比如background-color: lime，以进一步突显它。这样做是为了捕获那些有问题的标记出现的地方，无论这些标记是从CMS（Content Management System，内容管理系统）还是其他什么地方引入的。

你或许以为验证可以捕获任何标记问题，但其实不是。当然，当你使用了`font`元素时它会显示警告，但你可能会遇到验证器无法捕获的一些问题，我们来看一个普通的JavaScript链接示例：

```
<a href="#" onclick="javascript:nextPage();">Next</a>
```

这段代码在验证器下一切正常，因为标记是正确的。问题是，对于不使用JavaScript的用户，这个链接什么也不能做。这里应该有一个无脚本（non-JS）时的替代方案，并且应该提供一个可用的链接地址。因此，meyerweb的诊断样式表还有这样一行：

```
a[href="#"] {background: lime;}
```

这行代码会找出任何缺少无脚本后备链接地址的链接元素。（该代码使用了属性选择器，关于属性选择器详见2.11节。）

那么如何使用它诊断CSS呢？可以将它引入到你正在开发的网站的CSS中并在网站上线时删除，也可以把它设置成浏览器中的用户样式表，这样就可以在你访问的任何页面中使用了。

这里有一个完整的诊断样式表，它可以找出没有内容的空元素、没有`alt`或`title`属性或属性值为空的图像元素、找出没有摘要（`summary`）属性和表头含有无效范围（`scope`）值的表元素、含有有问题的或空的`title`和`href`属性值的链接元素等。注意，由于含有属性选择器，该版本无法在IE7中使用，而由于含有`:not()`和`:empty()`伪类，该版本也无法在IE8中使用。图1-32中展示了该诊断CSS的一个测试页。

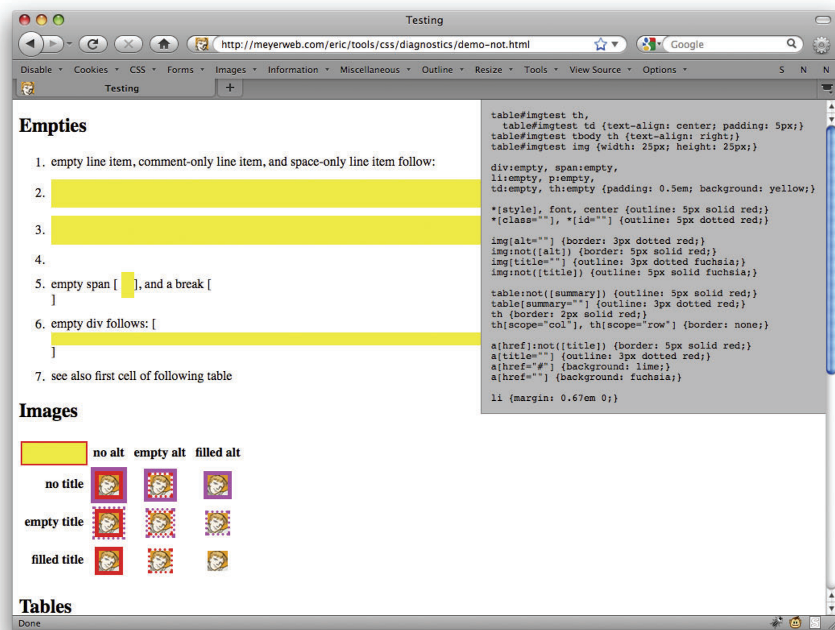


图1-32 诊断CSS的一个测试页

```
div:empty, span:empty,
li:empty, p:empty,
td:empty, th:empty {padding: 0.5em; background: yellow;}
*[style], font, center {outline: 5px solid red;}
*[class=""], *[id=""] {outline: 5px dotted red;}
img[alt=""] {border: 3px dotted red;}
img:not([alt]) {border: 5px solid red;}
img[title=""] {outline: 3px dotted fuchsia;}
img:not([title]) {outline: 5px solid fuchsia;}
table:not([summary]) {outline: 5px solid red;}
table[summary=""] {outline: 3px dotted red;}
th {border: 2px solid red;}
th[scope="col"], th[scope="row"] {border: none;}
a[href]:not([title]) {border: 5px solid red;}
a[title=""] {outline: 3px dotted red;}
a[href="#"] {background: lime;}
a[href=""] {background: fuchsia;}
```

1.9　重启样式表

有一件关于CSS的事情你可能没有考虑过，那就是即便你创建了HTML文档后一行CSS也不写，CSS也总是对文档起作用。实际上，有大量的CSS被应用在“未应用样式”的文档上（如图1-33所示），它们全部来自于浏览器本身。标题元素的默认大小和字体粗细、不同元素和不同行之间的间距、列表项前面的项目符号，甚至块元素和行内元素之间的区别全部是默认样式集决定的。

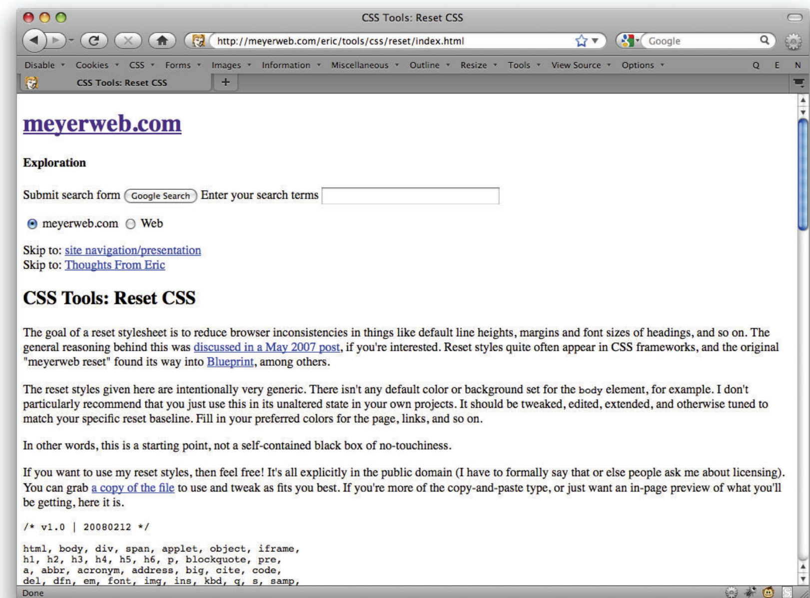


图1-33　实际上应用了许多样式的“未应用样式的”文档

当然，不同浏览器的默认样式略有不同，这不能怪浏览器厂商，因为还没有规范确切地说明文档应该具有什么样的默认样式。鉴于这种情况，大部分浏览器都尽量模仿Mosaic浏览器[①]显示文档的效果。是的，确实是Mosaic！因为它曾经是Netscape 1.0和IE3等浏览器试图模仿的对象。如果深入了解浏览器的默认样式，你就会发现那些从早期的Mosaic测试版浏览器复制过来的东西，甚至连像素值都是。

作为对策，很多人开发了重置样式（如图1-34所示），这意味着可以通过设置通用的属性尽量减少浏览器之间的不一致性，最简单的方式是：

```
* {margin: 0; padding: 0;}
```

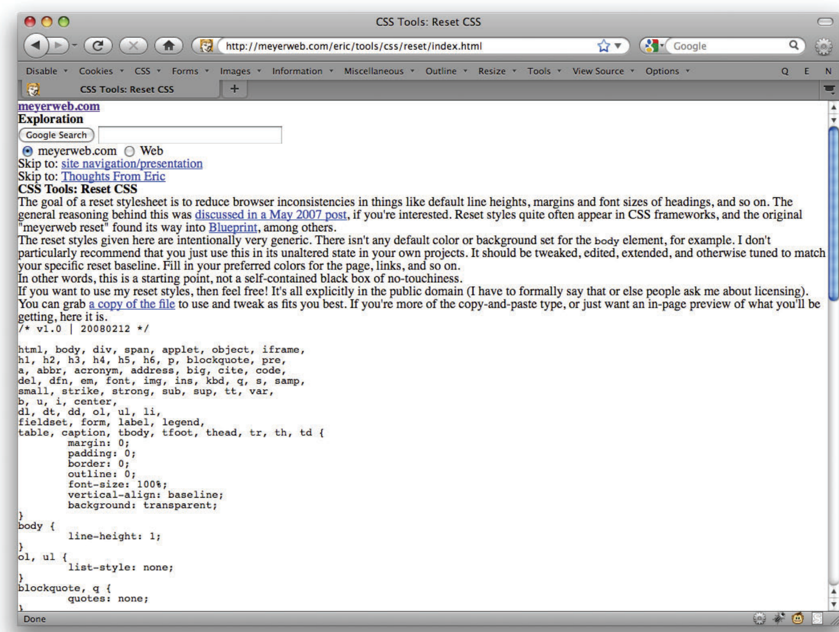


图1-34　应用了基本的重置CSS的文档

很多人都在用这行代码，因为它很简单。但问题是该规则会应用在文档中的全部元素上，包括像文本输入框和下拉选择框这种表单元素。由于目前不同浏览器处理表单元素样式的方法不尽相同（而且有一些根本不会应用表单样式），这种“全部元素”的方法就意味着在减少浏览器差异时表单元素的表现会变得很不一致。

正因如此，才有了更复杂一点儿的重置样式，其中一个比较流行的位于meyerweb.com/eric/tools/css/reset，它的开始部分如下所示：

---

① 互联网历史上第一个被普遍使用的浏览器，由美国伊利诺伊大学的NCSA组织在1993年发布，详情请到维基百科http://en.wikipedia.org/wiki/Mosaic_(web_browser) 查看。

```
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, font, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td {
    margin: 0;
    padding: 0;
    border: 0;
    outline: 0;
    font-size: 100%;
    vertical-align: baseline;
}
body {
    line-height: 1;
}
```

哇！元素可真够多的。之所以这么做是为了选择除表单元素（input、select、textarea）之外的所有元素并使它们表现一致。第一条规则明确地将外边距、内边距、边框和轮廓设置为0，并且强制将所有的元素设置成同样的字号和相同的文本垂直对齐方式。第二条规则为body元素设置了一个较小的行高，这个行高值会被body元素的所有后代元素继承。

在meyerweb的重置样式中还包含更多的规则，包括移除列表的项目符号和在blockquote（块引用）标记和q元素外侧自动生成的引号的规则等。再一次地，基本目的还是为了让我们在开始书写CSS之前，尽量使所有的浏览器表现一致些，同时也让页面好看一些。

此时你或许会想："等等，这就意味着我得撤销之前的全部工作！我从来也没想让页面有个一倍的行高啊！那就太局促了！我也不想让垂直对齐把上下标搞的乱七八糟！"

这些担心都是正常的，而你需要做的只是把样式表按照你的偏好进行修改。假设你经常使用1.4倍的行高来使文字显得疏落有致，只需要修改重置样式表中的这一行：

```
body {
    line-height: 1.4;
}
```

既然已经找到这儿了，除了自己喜欢的字体，你可能还想为页面添加一个你喜欢的背景色和统一的文本颜色：

```
body {
    font: smaller/1.4 Helvetica, sans-serif;
    background: #ABACAB;
    color: #444;
}
```

如果不想改变列表的默认样式，可以将清除列表项目符号的规则删除（但是我没有这么做，因为我们都有不同的需求）。你还可以继续添加精确定义列表的缩进方式和缩进量的规则、段落和列表项之间的间距规则以及你喜欢的加粗样式等。

至此，你拥有的已经不是重置样式表了，而是一个重启样式表（reboot style sheet）。通过此样式表可以将浏览器重启成你首选的修饰文档的基准样式，在此自定义的基点上可以建立任何项目。有了这个重启器（rebooter），任何项目都不用重新开始了，你可以把它作为核心的样式并在其基础上编写最终的样式表。

可以很负责地说，你不仅可以使用CSS重启浏览器样式，还可以用JavaScript来改良一些浏览器。

1.10　IE9.JS

Dean Edwards的IE9.js可以使IE5到IE8各版本的浏览器在处理CSS和HTML时更像IE9[①]（截至撰写本书之时IE9还没有发布）。你可以在code.google.com/p/ie7-js找到它，注意这里ie7并没有写错（如图1-35所示），因为这个项目是从IE7.js开始的，然后是IE8和IE9，新版本是很必要的。

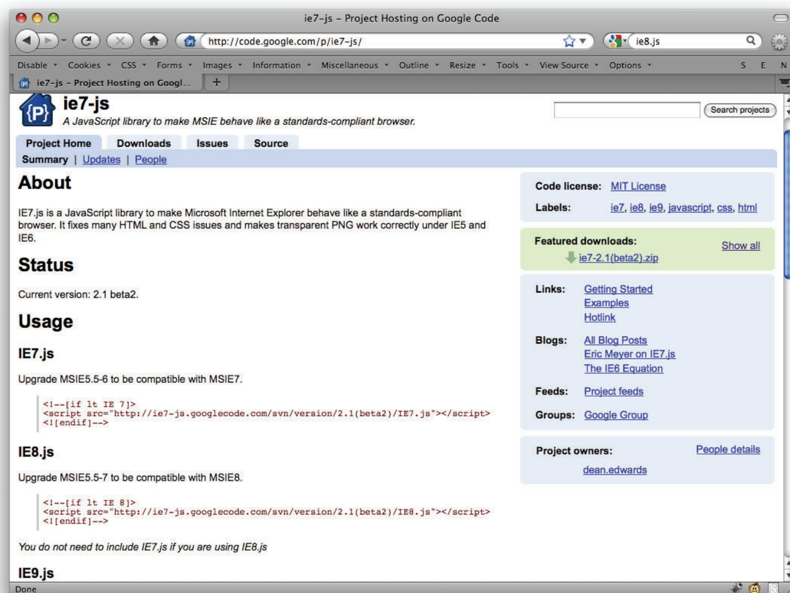


图1-35　IE7.js页面

IE9.js是一个JavaScript例程集，如果浏览器是IE9以下的版本，它就会扫描页面的CSS和HTML，并且计算出哪部分是当前IE版本不支持的，然后它会悄悄地在后台执行一番花哨杂耍式的操作，让浏览器支持这些原来并不支持的部分。

举个例子，IE5和IE6不支持属性选择器，假设我们有下面这样一条规则：

```
a[href] {text-decoration: none; color: red;}
```

① 在翻译本文时，微软已经发布了IE10平台预览版。

IE5和IE6会完全忽略这条规则，而链接元素不会有任何变化，尽管这可能让Jakob Neilsen[①]很高兴，但是该项目的设计者可能就没法给人留下深刻印象了。IE9.js花哨杂耍式的操作可以确保IE5和IE6能够将那些样式应用在链接元素上，使它可以正常工作。你需要做的只是将IE9.js链接到任何需要用到它的页面。

当然，如果禁用了JavaScript就什么效果都没有了，这意味着需要权衡一下，比如是否他们大部分还在使用IE6，以及他们有没有可能禁用JavaScript等。当然，这些是我们设计任何网站时都需要权衡的，也算是轻车熟路了。

通常的使用建议是将链接到JavaScript文件的script元素用一个条件注释包起来，像这样：

```
<!--[if lt IE 9]>
<script src="/code/IE9.js" type="text/javascript"></script>
<![endif]-->
```

脚本本身会确保只在需要的时候才运行，所以可以省略条件注释，然而这么做的话就意味着不管浏览器是否需要运行该脚本，每位访问者都会把它完整地下载下来。而有了条件注释，就可以确保只有那些有可能运行该脚本的浏览器才会加载该脚本。

如前所述，对于这一脚本还有一些早期的版本，它们可以使之前的旧版IE提升到像IE7或IE8那样，如果你觉得IE9.js不能满足你的需求，那就试试之前的版本吧。

① 权威的网站可用性顾问，详见http://en.wikipedia.org/wiki/Jakob_Nielsen_(usability_consultant)。

第 2 章 选　择　器

我们能真切地感觉到，选择器是CSS的核心部分。如果没有它们的话，我们除了把属性嵌入到每个元素里，就没有其他办法能把样式应用在元素上了，那真是太糟糕了。通过选择器赋予的选择任何形式任何种类元素的能力，我们可以只用很少的几行CSS完成很大一部分样式设置工作。

本章我们将深入探讨如何巧妙地使用选择器，并且概述一下哪些类型的选择器被普遍支持且应用最为广泛。

2.1 伪类与伪元素

在CSS中有两种“伪”字头的选择器：伪类（pseudo-class）和伪元素（pseudo-element）。CSS2.1中的伪类有：

- `:link`——未访问过的链接；
- `:visited`——访问过的链接；
- `:hover`——鼠标悬停的元素；
- `:focus`——获取焦点的元素；
- `:active`——激活的元素（例如一个被单击的链接元素）；
- `:first-child`——作为其他元素第一个子元素的元素；
- `:lang()`——根据元素的lang属性确定的元素。

CSS2.1中的伪元素有：

- `::first-line`
- `::first-letter`
- `::before`
- `::after`

那么区别在哪儿呢？区别就在于这些伪选择器影响文档的方式不同。伪类的表现有点儿像给文档添加类，而伪元素的效果就好像有元素被插入到了文档中。

以`::first-letter`为例，假设你要把每个`h1`的第一个字母增大到其他字母的两倍半（如图2-1所示），很简单：

```
h1::first-letter {font-size: 250%;}
```

这就仿佛CSS和标记被修改成了这样：

```
h1 first-letter {font-size: 250%;}

<h1><first-letter>H</first-letter>owdy, y'all!</h1>
```

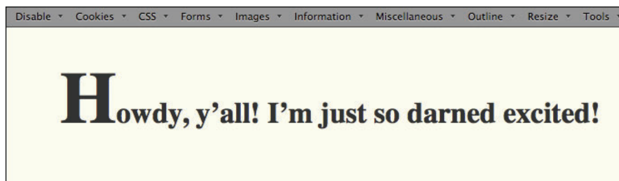


图2-1　放大`h1`的第一个字母

这真的是浏览器内部真实发生的情况吗？谁知道呢，反正结果确实像是发生了这种情况，因此才有了“伪元素”这个名字。

类似地，伪类的表现就像是它们使文档中的元素被添加了新的类。例如，想象一下若对于每一个作为其他元素第一个子元素的元素，浏览器给它们都附加了一个名为`first-child`的类，然后就可以像下面这样为它们应用样式了：

```
li.first-child {border-left: none;}
```

只需要简单地把点改成冒号就变成了`li:first-child`，就可以得到同样的最终效果，而不用费力把类添加到所有标记上。

有时还会看到伪元素使用双冒号的语法，这是在CSS2.1之后引入的。截至撰写本书之时，还没有哪个浏览器要求你必须在伪元素前面使用双冒号的，一个冒号就可以了。

另外提醒读者，CSS3增加了如下一些伪类，截至撰写本书之时，它们中的大部分还没有被广泛地支持：

- `:target`
- `:root`
- `:nth-child()`
- `:nth-of-type()`
- `:nth-last-of-type()`
- `:first-of-type`
- `:last-of-type`
- `:only-of-type`
- `:only-child`
- `:last-child`
- `:empty`
- `:not()`

- ❑ `:enabled`
- ❑ `:disabled`
- ❑ `:checked`

2.2　为目标元素添加样式

当希望指向文档中的某个片段时，目标（target）会非常有用。什么，怎么实现？其实非常简单：

```
<a href="http://example.com/law.html#sec2-7">Section 2.7</a>
```

任何人单击这个链接（如果浏览器处理正确的话）将不止跳转到目标页，而且还会跳到页面中文档片段标识符（地址中的`#sec2-7`部分）出现的地方。这有时是通过锚点（anchor）实现的，但是只用ID来实现会更好一些。例如有如下两种场景：

```
<h3><a name="sec2-7">Exceptions</a></h3>
<h3 id="sec2-7">Exceptions</h3>
```

这两种情况下，当浏览器跳到了文档中的目标位置时，都不会有任何视觉提示告知你已经到达了目标位置，而使用`:target`伪类就可以给出视觉提示。例如，若想让作为某个文档片段标识符目标的`h3`拥有特定的提示效果（如图2-2所示），可以这样写：

```
h3:target {color: maroon;
    background: #FFA;}
```

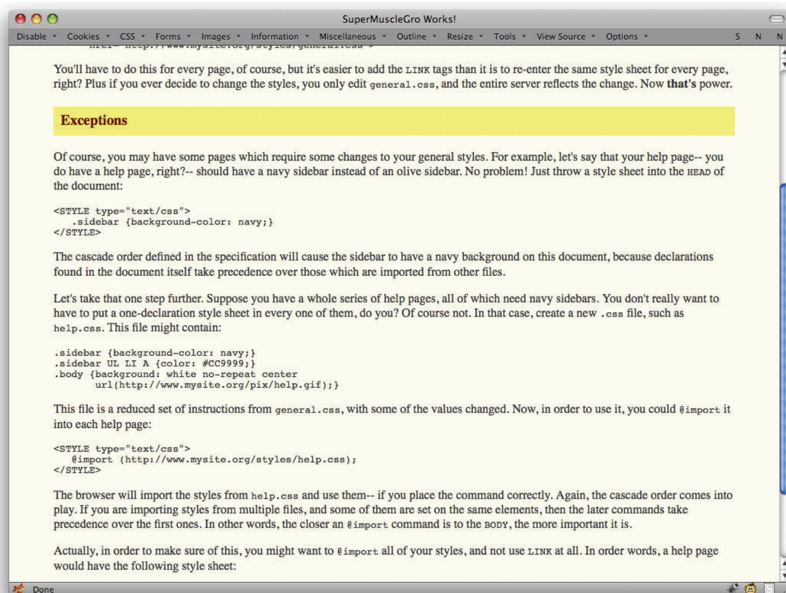


图2-2　突出显示目标元素（另见彩插图2-2）

当然，你或许想把这个样式应用到任何目标元素上，而不管它是什么元素，那么只需要把h3换成一个通用选择器即可，就像这样：

```
*:target {color: maroon;
    background: #FFA;}
```

从技术上讲，这种情况下通用选择器也是可选的，可以把这个选择器简单地写成:target。

如果想让目标样式多一点Web 2.0的感觉，还可以设置一个渐隐背景的效果。你懂的，就是那种“你已经完成了某项操作，所以页面上的一块背景会从黄色变到白色，让你知道已经完成了该项操作”的效果。通过:target和一个动画GIF可以很容易地实现这种效果，只需要创建一个从黄色渐变到白色（如果白色是你网站的背景色的话）的动画并且把它当做背景图像。

```
*:target {background: url(/pix/yellow-fade.gif);}
```

2.3　特殊性

你很难快速地把特殊性（specificity）这个词读3遍，而若想彻底地掌握它甚至更难。但是，它却是理解CSS规则之间相互作用的关键。

特殊性是一个选择器“特殊程度”的数字表示，有3样东西经常被用来确定选择器的特殊性：

- 每个元素描述符贡献0,0,0,1；
- 每个类、伪类或者属性描述符贡献0,0,1,0；
- 每个ID描述符贡献0,1,0,0。

先不要抓狂，来看几个小例子。

div ul ul li	0,0,0,4	4个元素描述符
div.aside ul li	0,0,1,3	1个类描述符，3个元素描述符
a:hover	0,0,1,1	1个伪类描述符，1个元素描述符
div.navlinks a:hover	0,0,2,2	1个伪类描述符，1个类描述符，2个元素描述符
#title em	0,1,0,1	1个ID描述符，1个元素描述符
h1#title em	0,1,0,2	1个ID描述符，2个元素描述符

希望这些能够帮助你理解特殊性是如何计算的。那么，为什么是逗号呢？因为可以这么说，每个“级别”的特殊性的值都是相互独立的。因此，具有一个单独的类描述符的选择器会比由13个元素描述符组成的选择器拥有更高的特殊性。

```
.aside    /* 0,0,1,0 */
div table tbody tr td div ul li ol li ul li pre    /* 0,0,0,13 */
```

第一个选择器左数第三位的“1”胜过了第二个选择器同样位置的“0”，基于这样的事实，第二个选择器第四位的“13”（在这个非常有限的例子中）就毫无意义了。逗号可以使我们看得更清楚，否则选择器的特殊性可能被写成“10”和“13”，从而造成后者特殊性更高的假象（在CSS的早期还没有引入逗号分隔符时，这是经常出现的误解）。

还有另外一种常见的误解，就是特殊性的结构相近问题。例如，假设有如下两个选择器：

```
ul li {font-style: normal;}
html li {font-style: italic;}
```

它们当中哪个会赢呢？它们都有两个元素描述符，这意味着它们的特殊性都是0,0,0,2。其实，后写的会赢，与html元素相比ul元素在文档中的位置离li元素更近也不管用。特殊性只是单纯的数值，它不会以任何方式评估页面的结构。结果，列表元素将全部变成斜体，因为当特殊性相等时后声明的规则会胜出。

因为我说过有3样东西影响特殊性，所以你或许想知道特殊性标识符第一位的0是干嘛用的。其实，第一个0是用于行内样式（inline style）的，且仅用于行内样式。因此，如果有下面这样的样式和标记，则div的背景将会是蓝色。

```
div#header {background: purple;}  /* 0,1,0,0 */

<div id="header" style="background: blue;">  <!-- 1,0,0,0 -->
```

2.4 重要性

有一样东西是可以无视特殊性的，那就是!important。如果你是一名程序员的话，我现在就要打消你的错误想法——!important不代表“不重要”[①]。

可以使用!important把任何单独的声明标记为重要。下面是一个简单的例子：

```
a:hover {color: red !important; text-decoration: none;}
```

在这个例子中color: red被标记为重要，但是text-decoration: none没有。任何你想标记为重要的声明都需要有自己的!important。

基本上，任何重要的声明都会覆盖非重要的声明。好了，就此打住吧。使用下面的代码，结果将会得到一个绿色的链接：

```
div#gohome a#home {color: red;}
div a {color: green !important;}

<div id="gohome"><a id="home" href="/">Home</a></div>
```

第一个规则非常高的特殊性（0,2,0,2）对于解决颜色冲突没有任何作用，因为!important可以胜过它。

当然，如果我们为第一条规则也添加一个重要标志，那么情况就不同了。

```
div#gohome a#home {color: red !important;}
div a {color: green !important;}
```

由于两个颜色声明都是重要的，所以会采用正常的层叠规则来解决冲突，换句话说，特殊性又起作用了，所以链接会变成红色。

这提醒我们使用!important时要非常小心，因为一旦开始用它覆盖其他规则，很快你就会

① 多数编程语言里叹号都有取否定的意思，所以作者说!important并不是不重要。

发现必须再用另外的!important覆盖这条重要的规则，因此就必须再引入一个!important声明，最终会把所有的声明都变成重要的，也就意味着最终没有一个规则是重要的。

2.5　省略简写属性值的关键词时会发生什么

我们都知道CSS中有很多简写属性，其中background、border、font、margin和padding是几个最常用的，这是一次表达一堆信息的高效而简洁的方式。但是，当省略了属性值中的某些部分时会发生什么呢？考虑下面的代码：

```
strong {font: bold italic small-caps medium/1.2 Verdana, sans-serif;}
```

如图2-3所示，这会对strong元素应用1.2倍行高且字体为粗斜体、中等字号、小型大写字母（small-caps[①] medium-size）的Verdana（或者其他sans-serif字体族[②]的字体）样式。

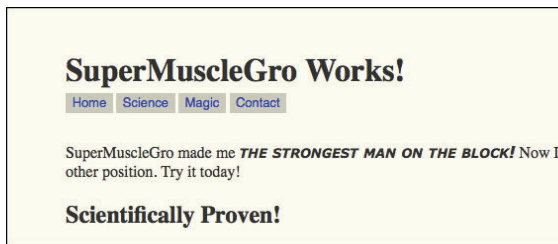


图2-3　疯狂的strong

假设我们缩减一下这个属性值：

```
strong {font: medium Verdana, sans-serif;}
```

则结果是中等大小、正常粗细的Verdana字体（如果Verdana字体不可用的话就是其他的sans-serif字体），而粗体的效果消失了，如图2-4所示。

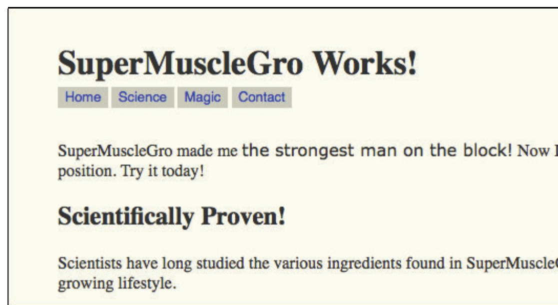


图2-4　因疏忽而无加粗效果

原因是当省略了部分简写属性值的时候，缺失的部分就会使用该属性的默认值。因此，当省

① 小型大写字母，参见http://en.wikipedia.org/wiki/Small_caps。
② 无衬线字体，参见http://en.wikipedia.org/wiki/Sans-serif。

略了字体粗细（font-weight）、字体样式（font-style）和字体异体（font-variant）时，实际相当于这样：

```
strong {font: normal normal normal small/normal Verdana, sans-serif;}
```

是的，即使行高line-height也有默认值，该默认值可以覆盖任何继承的行高值。

如果设置样式的时候不小心，就可能会出现类似的问题。考虑下面两条规则，第一条来自于全站的样式表，而第二条来自于某个页面的嵌入样式。

```
body {background: #FCC url(/i/pagebg.gif) 10px 25% no-repeat fixed;}
body {background: url(i/body-bg.gif);}
```

在这两条规则的作用下，这里讨论的页面会有一个新的从左上角开始的背景图像，且当页面滚动时会随着页面滚动。这是因为第二条规则实际上等价于：

```
body {background: transparent url(i/body-bg.gif) 0 0 repeat scroll;}
```

现在，如果这确实是想要的效果，那么你可以这么做，但它的目的似乎更像是要把图片换成另外的图片。其实在这种情况下，你只需要设置特定的属性，像这样：

```
body {background-image: url(i/body-bg.gif);}
```

总之，这是大部分简写属性的工作方式。不过，也有一些例外的，如margin、padding、border-style、border-width和border-color等，省略这些属性的值，结果就像是从已给出的值“复制”了一份。下面有几个功能相同的声明：

```
margin: 1em;    margin: 1em 1em 1em 1em;
padding: 10px 25px;    padding: 10px 25px 10px 25px;
border-color: red green blue;    border-color: red green blue green;
```

当然，这些值的顺序还是上右下左（Top-Right-Bottom-Left），可以简记为TRBL，这可以免除你“记不住”这一麻烦（TRouBLe）。

2.6 选择性地覆盖简写属性

虽然简写属性会用默认值覆盖未声明的部分，但这并不意味着我们需要避免这种情况。实际上，你可以用简写属性声明80%，然后在另一处通过覆盖实现另外20%。

假设你想实现一个具有3 px的点线边框，其中3条边是黑色，而第四条边是红色（如图2-5所示），可以每次都写一个边，但是那样要多敲很多次键盘。所以可以像下面这样声明：

```
border: 3px dotted black;
border-left-color: red;
```

这样，就只需要调整那一小部分跟其他部分不同的地方。你甚至可以用同样的规则实现一些更好的效果。

再举一个常见的选择性覆盖简写属性的例子。注意，标题元素可能除了字号以外大部分属性都相同。如果你对浏览器默认的字号很满意的话，就可以直接这么写了：

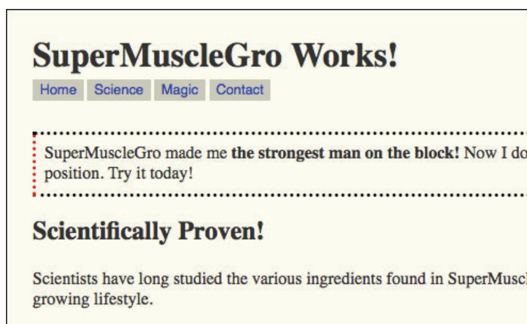


图2-5　把一侧的边变成红色（另见彩插图2-5）

```
h1, h2, h3, h4, h5, h6 {font-weight: normal;
font-style: italic;
font-family: Helvetica, sans-serif;
line-height: 1.5;}
```

另一方面，如果想设置自己的标题字号，如图2-6所示，则需要换个思路：

```
h1, h2, h3, h4, h5, h6 {font: italic 100%/1.5 Helvetica, sans-serif;}
h1 {font-size: 225%;}
h2 {font-size: 185%;}
h3 {font-size: 140%;}
/* …… */
```

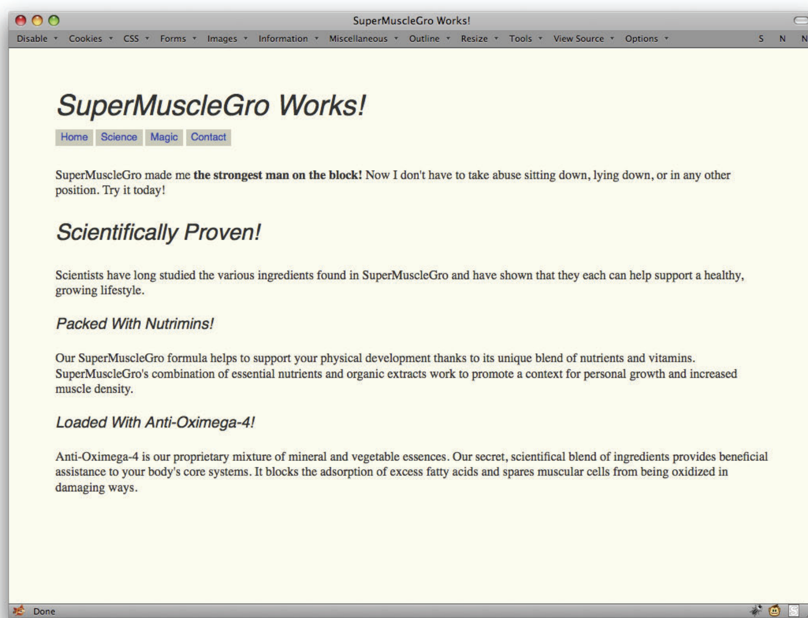


图2-6　通过选择性覆盖快速设置标题字号

当使用这种选择性覆盖时，最好确保用来覆盖简写属性的属性出现在简写属性之后。只有通过这种方式，当选择器具有相同的特殊性时（这也是经常出现的情况），用来覆盖简写属性的属性才能胜过相应的简写属性。

2.7　通用选择

2

本节我将介绍一下选择器中星号（*）的使用，先别太兴奋，它并不是你想象中的万能牌。下面是一个简单的例子：

```
* {color: blue;}
```

这个星号称为通用选择器，该选择器的作用是选择文档中的全部元素并对其应用样式。

这看起来像是一个通配符，而且在某种情况下确实是，因为你可以用它选择一大堆元素而无需给它们命名。假设我想选择这个div中的全部元素：

```
<div>
<h1>Hey-ho!</h1>
<p>I'm a <em>paragraph</em>.</p>
<ol>
<li>Uno</li>
<li>Deux</li>
<li>Drei</li>
</ol>
</div>
```

就这么简单：

```
div * {border: 1px solid red;}
```

结果跟这样写的效果是一样的：

```
div h1, div p, div em, div ol, div li {border: 1px solid red;}
```

好吧，应该说是几乎一模一样。如图2-7所示，视觉效果是一样的，但是还有一个非常轻微的不同，那就是它们的特殊性。你可以看到，通用选择器的特殊性贡献为0,0,0,0，这意味着`div *`的特殊性为0,0,0,1，而`div h1`（还有这一组里的其他选择器）的特殊性为0,0,0,2。除了这个，结果是一样的。

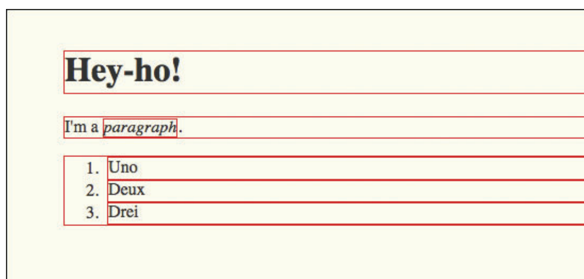


图2-7　把div的所有后代加上红框（另见彩插图2-7）

你或许以为可以用h*来代替h1、h2、h3、h4、h5、h6，但是对不起，不行。就像前面展示过的，只能把它作为一个元素通配符，这就是它的全部功能了。

2.8　ID 还是类

任何一个有抱负的网页设计师遇到的第一个大难题就是：我该用类（class）还是ID呢？

就像生活中的许多事情一样，这个问题有一个简单的答案，然而也有一个非常复杂的答案。简单的答案是：对于任何在页面中可能出现不止一次的“标签”使用类，对于任何只出现一次的使用ID。这里所谓的“标签”，指的是一个附加到元素上的描述性词汇。实际上，类和ID有99.44%的可能性被用在描述元素上。

使用ID值的两个经典例子是页头（header）和页脚（footer），前提是任何给定页面都只有一个页头和一个页脚。类值的应用就比较分散了，比如可以用more表示“详细信息”链接，用tabs表示一组导航选项卡，或者用odd表示奇数表格行。

至于那个更复杂的答案就不仅要权衡标签是否在页面中唯一了，还要考虑到ID和类对特殊性的影响。由于包含ID的选择器比那些只包含类的选择器具有更高的特殊性，因此你可能会无法覆盖某个特定规则。

这里有个简单的例子，假设你已经在全站样式表中写过如下规则：

```
#header {background: black;}
#header a {color: white;}
```

然后你又决定让联系信息页面不要那么严肃，所以想把页头设置成浅灰色，并让所有的导航链接有一个舒缓的暗绿色样式。由于该联系信息页面有几组不同的导航链接，所以你会这么写：

```
#header {background: #BBB;}
.navlinks a {color: #257000;}
```

很遗憾，如图2-8所示，因为#header a具有更高的特殊性，所以页头的导航链接仍然是白色。

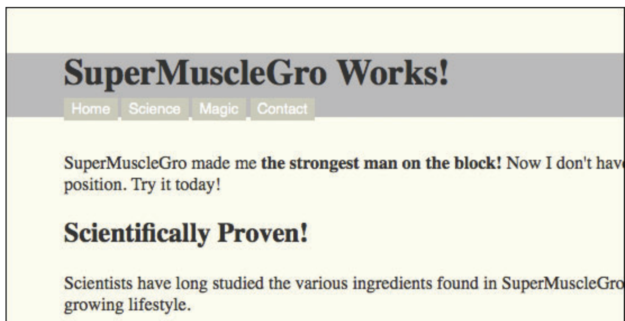


图2-8　页头毫无吸引力的链接

对于这个问题，可以这样来解决：

```
#header a, .navlinks a {color: #257000;}
```

甚至这样：

```
#header .navlinks a, .navlinks a {color: #257000;}
```

这两种方式都可以，不过看起来有点儿笨拙，不是吗？（虽然不如往`.navlinks a`上砸个`!important`那么笨，但还是有点儿笨。）另外一个处理这种情况的方法是将包含ID的页头标签里面的`id`换成类，即：

```
<div class="header">
```

此处的`class="header"`原来是`id="header"`。这样就不用太担心ID弄出来的那些很难搞的特殊性冲突问题了。也就是说，在你的全站样式表中这么写：

```
.header {background: black;}
.header a {color: white;}
```

然后在联系信息页面的样式表中这么写：

```
.header {background: #BBB;}
.navlinks a {color: #257000;}
```

最终结果就是漂亮的绿色文字链接，如图2-9所示。

图2-9　页头引人注目的链接（另见彩插图2-9）

这就是全部需要做的工作了，所以，这是为大部分或者全部标签都应用类的一个好理由。

另外一个理由是，你没法确定什么时候标签会从一个变成多个。页头（header）就是个最好的例子，因为一个页面中可能有多个"页头"。如果觉得奇怪的话，想想新闻网站或者其他门户网站，每个部分和侧栏框都会有自己的"页头"，并且页脚也一样。因此一贯地使用类就很合理了。

现在，你或许会争辩说那些并不是一个页面上真正的页头和页脚，它们只是标题和其他附加信息或者诸如此类的内容。其实这是个语义问题，是没法明确解决的，比如你称为首行的，我可能叫它头部。关键是你用于网页的唯一标签有朝一日可能变得不再唯一。避免这种情况发生的最好办法就是一开始就全部使用类。

那么有没有什么用得着ID的地方呢？当然可以有。总会有一些情况下，你可以确定某个元素在页面中是唯一的，并且永远不会重复。还有一些情况下，你希望用ID给选择器赋予更高的特殊性，以便轻松地使它胜过其他选择器。并且ID对于脚本编程、链接目标和其他CSS之外的东西都

至关重要，你只需要在书写CSS时小心一点儿就是了。

还有另外一种可以使用ID且不用担心特殊性问题的方法，我们在2.13节介绍。

2.9　ID与类共用

偶尔可能会有这样的情况，有一个唯一的元素并且它还是一大类元素中的一分子。例如，假设站点侧边栏中有一堆小面板，每个面板都有一个框并且具有特定的颜色和字体，但是每个面板都有独特的地方，比如每个都有不同的背景图等。

在这种情况下可以给元素同时应用类和ID，就像这样：

```
<div class="panel" id="weather">
<div class="panel" id="stocks">
<div class="panel" id="latest">
```

然后，在CSS里就可以按需处理了。

```
.panel {
    border: 1px solid silver;
    background: #EEE top left no-repeat;
    color: #333;
    font: x-small sans-serif;}
#weather {
    background-image: url(/pix/panel-weather.jpg);}
#stocks {
    background-image: url(/pix/panel-stocks.jpg);}
#latest {
    background-image: url(/pix/panel-latest.jpg);}
```

你甚至可以把其中两个组合为一个选择器，像这样：

```
.panel#weather {font-weight: bold;}
#latest.panel {color: #511;}
```

没错，它们的书写顺序无关紧要，不必跟它们在HTML中出现的顺序一致。

2.10　多类

元素的`class`属性有个经常被忽略的能力，就是你可以用任意多的词组成一个由空格分隔的列表，将其作为该属性的值。换句话说，就是你可以给元素应用多个类。

作为例子，我们继续使用之前的标记并把它改成不使用`id`属性的状态：

```
<div class="panel weather">
<div class="panel stocks">
<div class="panel latest">
```

然后只需要调整CSS使其能够应对用类代替ID的情况：

```
.panel {
    border: 1px solid silver;
    background: #EEE top left no-repeat;
```

```
        color: #333;
        font: x-small sans-serif;}
    .weather {
        background-image: url(/pix/panel-weather.jpg);}
    .stocks {
        background-image: url(/pix/panel-stocks.jpg);}
    .latest {
        background-image: url(/pix/panel-latest.jpg);}

    .panel.weather {font-weight: bold;}
    .latest.panel {color: #511;}
```


类在HTML源代码中的书写顺序与在样式表中的书写顺序无关，即.panel.weather的效果跟.weather.panel的效果是一样的，并且无论这两个类在HTML源代码中的书写顺序如何，特殊性也都是一样的。它们在源代码中是否被其他类名分隔开也是无关紧要的，比如：

```
<div class="weather alert tornado watch panel">
```

此时通过.panel.weather和.weather.panel都可以选中该元素。

有个正在慢慢被忽略的注意事项[①]：IE6及早期版本不识别样式表中多个类的写法，当使用.panel.weather时，IE6只会识别成.weather。尽管在HTML中仍然可以使用多个类，并且可以在CSS中定位这些类，但是每次只能使用一个类。因此，.weather和.panel都可以在IE6下很好地匹配之前例子中的标记，只是IE6会假定.weather.panel给元素应用了一个包含panel这个词的类，这可能并不是你想要的结果。

2.11　简单的属性选择

属性选择器是CSS2中引入的并且在CSS3中得到了扩展，截至撰写本书之时已经得到所有主流浏览器的支持（IE6除外，如果这对你来说是个问题的话，参见1.10节）。

最基本的思路就是可以通过元素已有的属性选择元素，或者基于元素属性值的某个方面进行选择。因此，你可以选择所有确实是链接的a元素，像这样：

```
a[href]
```

该选择器会选择所有含有href属性的a元素，不会选择没有href属性的a元素，命名锚点（例如<a name="top">）就是最明显的例子。它基本上是a:link, a:visited的一个简化版。例如应用：

```
a[href] {color: green;}
```

则页面将会如图2-10所示。

至于href的属性值是什么一点都不重要，实际上甚至连属性值是否为合法的URI（Uniform Resource Identifier，统一资源标识符）或其他资源都无关紧要。选择<a href="#">和选择<a href="http://w3.org/">的方式完全相同。

① 虽然国外可以慢慢不去顾及IE6，但是在国内仍然是个标准的注意事项。

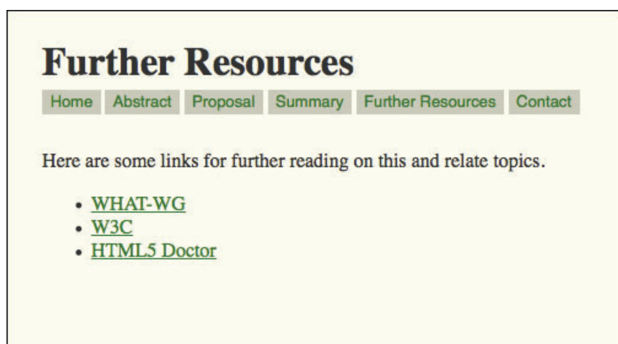


图2-10　通过属性选择器选择链接元素

现在，如果想选择指向某个特定地址的全部超链接该怎么做呢？若想筛选出一个特定的URI，只需要这么做（如图2-11所示）：

```
a[href="http://w3.org/"] {font-style: italic;}
```

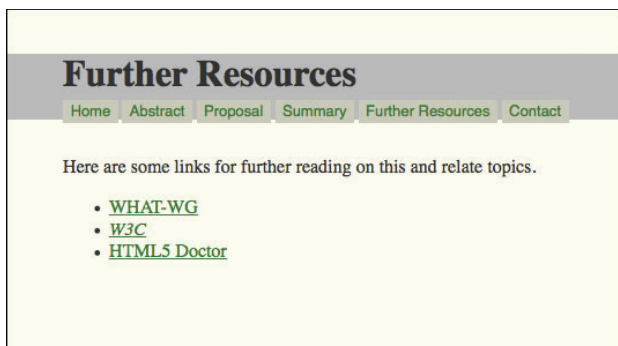


图2-11　通过属性选择器选择具有特定URL的链接元素

该选择器只会选择href属性值中包含http://w3.org/的链接元素，注意到我的措辞了吗？我没有说“指向W3C网站的超链接元素”，因为事实并非如此。这里面的原则是必须做到精确匹配，每个字母都要匹配。如果有这样一个链接<a href="http://www.w3.org/">，则该选择器不会选择这个链接，因为必须做到精确匹配才行。

或许这对于超链接来说用处不太大，但是它可以帮助筛选特定的图像（以便为其添加样式），比如公司的标志。如果你的CMS总是为页顶的标志生成这样的标记：

```
<img src="/img/2010/mainlogo.png" alt="ConHugeCo Inc." />
```

则总是可以用这样的方式选择该图像：

```
img[src="/img/2010/mainlogo.png"]
```

你不需要给它设置类或者ID或者其他什么东西，而只需要基于src的值给它设置样式。前提

是，正如我所说，你应该确定它总是会保持那个特定的值，而不会变成其他值。（对于非精确匹配，请参见2.14节。）

有一点需要注意，按照CSS规范，“选择器中属性名和属性值是否区分大小写取决于文档语言”（参见www.w3.org/TR/CSS2/ selector.html#matching-attrs）。换句话说，有些标记语言可能会区分属性名的大小写，而另外一些语言则不会。XHTML就区分大小写，所以你最好假定属性名和属性值都是区分大小写的。

2.12　类的属性选择

如果读过前面一节，你可能会想：“嘿，我可以用属性选择器改造一下.class符号啊！”是的，确实可以。只不过并不是我之前展示过的某种方法。

通过属性选择器我们可以实现跟div.panel一样的效果：

```
div[class~="panel"]
```

看到波浪号（~）了吗？就在等号的前面，在这种情况下它绝对是个关键部分。它的存在意味着该属性选择器会选择“以空格分隔的类名列表中包含该词的元素”，这就是小小波浪号的巨大作用。

为了加深理解，我来展示一下没有波浪号的话会发生什么。去掉波浪号后的选择器为：

```
div[class="panel"]
```

这会选择任何class属性的值为panel且只能为panel的div元素，如果某个元素的class属性值为panel weather，则这个选择器无法匹配该元素，因为panel和panel weather不是精确匹配。而另一方面，div.panel则可以很好地匹配<div class="panel weather">。

通过使用波浪号可以获得与“点类”（dot-class）语法完全相同的效果，因此下面两个规则除了输入的字母不同之外，其他方面完全相同：

```
div[class~="panel"]
div.panel
```

这时候你或许会想，“嘿，真棒！我一直想知道如何才能选择一个又长又复杂的类呢。”嗯，但是记住，属性选择器并不只局限于我们上面使用过的这两个属性——类和id。你可以选择任何属性值为可以被空格分隔的一串单词的元素，而说到“单词”，我指的是“字符构成的字符串”。

下面是使用属性选择器的其他几个例子：

`img[alt~="figure"]`	任何alt属性文本中包含“figure”的图像元素
`table[summary~="data"]`	任何summary属性文本中包含“data”的表格元素
`*[title~="2009"]`	任何title属性文本中包含“2009”的元素

2.13　ID 还是属性选择器

不仅可以把属性选择器当做类选择器的加长替代版，还可以用它代替ID选择器。下面两条规则会选择同样的元素：

```
p#lead-in {font-weight: bold;}
p[id="lead-in"] {font-weight: normal; font-style: italic;}
```

很好，不过先花点儿时间考虑一下这两条规则的视觉效果：如图2-12所示，lead-in段落会同时拥有加粗和斜体的效果。

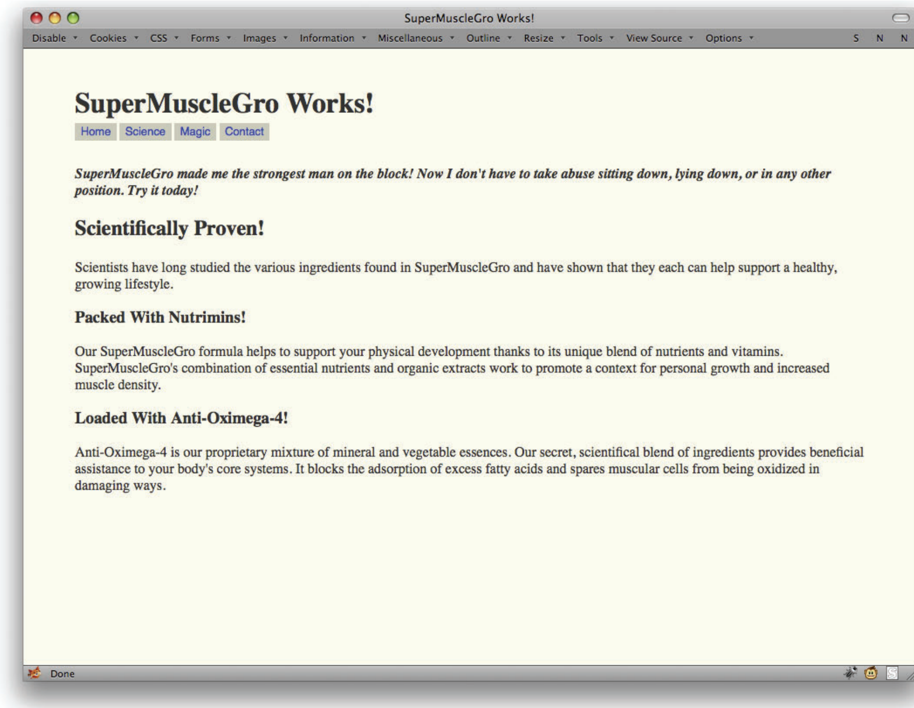


图2-12　根据不同的特殊性合并样式

这是因为属性选择器的特殊性贡献为0,0,1,0，与类和伪类的贡献相同。因此第一条规则的特殊性为0,1,0,1，而第二条规则的特殊性为0,0,1,1。在字体粗细的斗争中，第一条规则因其较高的特殊性胜出了。

这是特殊性的一个有趣的小技巧，通过它可以引入崭新的创作模式。例如，你或许还记得之前在2.8节讨论过的ID会轻松地胜过类，并且你或许已经考虑用类来选择所有的标签了。如果用户群都在使用支持属性选择器的浏览器，那么你就又可以把ID和类混着用了，然后在需要通过ID引用元素时使用属性选择器即可。通过这种方式，就不用担心#ID选择器的特殊性会胜过你写的其他任何东西了。

2.14 部分属性值选择

当CSS2完成之后，下个版本的CSS制定工作就开始了，即我们称为CSS3，尽管已经不再是一个规范了（这个说来话长）。其中一个最受关注的领域就是选择器，而属性选择器也不例外。规范的制定者们挑选出了一系列子串匹配模式，它们全都非常有用。

最基本的一个是子串匹配器。要想看看它有多大用处的话，考虑一个之前的例子：

```
a[href="http://w3.org/"]
```

该选择器会选择全部指向这个精确URL的链接元素，这没有任何问题。然而，假设有很多个链接到W3C网站，而不只是主页的链接，并且你还想对它们应用同样的样式。解决这个问题的一个好办法就是只选择URL中的w3.org部分，如图2-13所示。下面是具体做法：

```
a[href*="w3.org"] {font-weight: bold;}
```

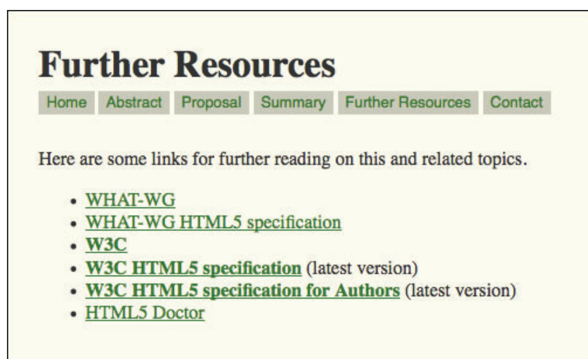


图2-13 选择所有URL中包含w3.org的链接元素

是的，只需要在等号前面加个星号就行了。注意这不是通用选择器，也不能把星号放到值的前面来创建UNIX或者grep式的通配符。你只能把它放到等号的前面，意思是“属性值中将包含该字符序列”。

一如既往，这个选择器也可以用在任何元素的任何属性上。回顾之前选择公司唯一标志图像的例子，你还可以这样写：

```
img[src*="mainlogo.png"]
```

它会选择任何指向mainlogo.png文件的图像（`img`）元素，或者是`src`属性值中包含`mainlogo.png`这些字符的图像元素。因此，它将同时选择：

```
<img src="/img/2010/mainlogo.png.gif" alt="ConHugeCo Inc." />
<img src="/img/2010/mainlogo.pngdir/big.png" alt="ConHugeCo Inc." />
```

然而，你或许不应该这样命名文件和目录。当然，这只是建议。

有许多创造性的方式可以使用这种特殊的能力，通过选择图像元素URL中对应的部分可以选择来自于特定目录中的图像。因此，根据链接元素的`href`属性值，你就可以对链接到你网站某

个特定区域的链接应用样式了。

```
a[href*="/contact"] {color: maroon;}
a[href*="/news"] {font-weight: bold;}
```

请牢记，应该区分属性值的大小写，因为这样可以把事情变得简单。因而下面的3个例子中头两个可以匹配到，而第三个则不能。

```
img[alt*="Figure"] {border: 1px solid gray;}

<img src="fig1.gif" alt="Figure 1. The larch." />
<img src="fig2.gif" alt="Figure 2. Mayor Quimby, a political figure of some note." />
<img src="digg.gif" alt="Several men trying to figure out how to dig a hole." />
```

第三个图像之所以没有匹配是因为“figure”与“Figure”是不同的。当然，在这种情况下或许是个好事，因为（根据alt的文本）第三个图像并不是作为常规的插图（Figure）出现的，它只是恰好在alt属性值中含有“figure”这个词而已。这也没关系，但是要知道下面的情况也会被匹配：

```
<img src="lost.gif" alt="Lost again. Figures, don't it?" />
```

是的，是一样的“Figure”，所以会匹配。

如果知道大小写只差一个字母，那就很容易突破这种限制了。因此，如果想确保选择全部包含“Figure”和“figure”的实例，那么可以让选择器变成这样：

```
img[alt*="igure"] {border: 1px solid gray;}
```

当然它也会匹配诸如“configure”、“disfigure”和“oliguresis”等（仅举几例）这样的实例。

然而，子串选择并不止于此。具体将在下一节详细解释。

2.15 更多部分属性值选择

尽管任意属性值的子串匹配已经很好了（见2.14节），但有时你可能想限制只查找属性值的开头或结尾部分。幸运的是，确实有几个办法可以实现。

如果想根据属性值的开始部分的子串进行选择，可以使用这种模式：

```
a[href^="http"]
```

由于有了脱字符（^），该规则会选取任何href属性值是以http开头的链接元素。用这种方式很容易选择全部的站外链接。假设全部站内链接都是相对于页面或者站点的，并且在文件系统里不使用http字符串，则你可以简单地这样写：

```
a[href^="http"] {font-weight: bold;}
```

或者更复杂一点儿，像这样：

```
a[href^="http"] {padding-right: 18px;
    background: url(/pix/external.png) 100% 50% no-repeat;}
```

结果如图2-14所示。

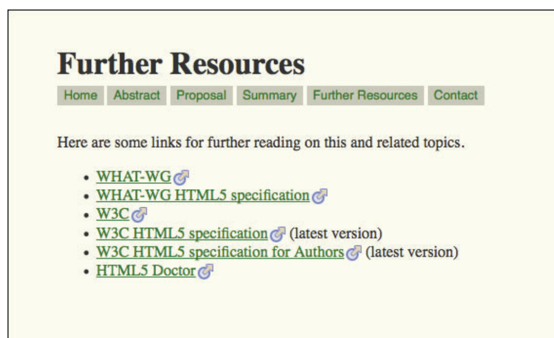


图2-14　为http开头的链接添加图标

要想根据属性值的结尾子串选择元素，可以使用这个模式：

```
a[href$=".pdf"]
```

由于有了美元符号（$），该规则会选择href属性值是以.pdf结尾的链接元素。这是使PDF文件下载链接变得显眼一些的简单办法，如图2-15所示。例如：

```
a[href$=".pdf"] {padding-right: 18px;
    background: url(/pix/pdf.png) 100% 50% no-repeat;}
```

图2-15　PDF文档链接的PDF图标

这真是太棒了！下面还有其他一些使用属性选择器对链接应用样式的例子。

`a[href^="https"]`	安全服务器链接
`a[href^="mailto"]`	电子邮件联系链接
`a[href^="aim"]`	AOL即时通信服务链接
`a[href$=".doc"]`	微软Word文档
`a[href$=".xls"]`	微软Excel文档
`a[href$=".zip"]`	压缩文档

一如既往，记住这里并不局限于超链接。如果回想一下2.14节里“Figure”的那个例子，你会很快意识到许多问题都可以用一个简单的脱字符解决：

```
img[alt^="Figure"] {border: 1px solid gray;}
```

现在我们选择的图像元素的alt文本是精确地以“Figure”开始的，并且不用担心“Figure”出现在alt文本的中间或者其他地方的情况，这些情况都会被跳过。

2.16　选择后代元素

基于元素在文档结构中的位置选择元素也很常见，这通常是使用后代选择器实现的，就像这样：

```
div#header a {color: #DEFACE;}
```

该规则会选择id为header的div的后代元素（包含在div中）中的全部锚点元素（a）。

大多数情况下，这确实是我们想要的效果：选择头部的链接元素而不用管它们在头部的位置，也不用考虑两个元素之间会不会有其他元素。

然而，有时候你或许想选择作为其他元素子元素的那些元素，而不是任意的后代元素。想象一下只选择作为有序列表（ol）元素子元素（而不是后代元素）的列表项（li），如图2-16所示。这样，如果有任何无序列表包含在该有序列表中，则它们的列表项不会被选中。我们需要的只是一个子选择符。

```
ol > li {list-style-type: upper-alpha;}
```

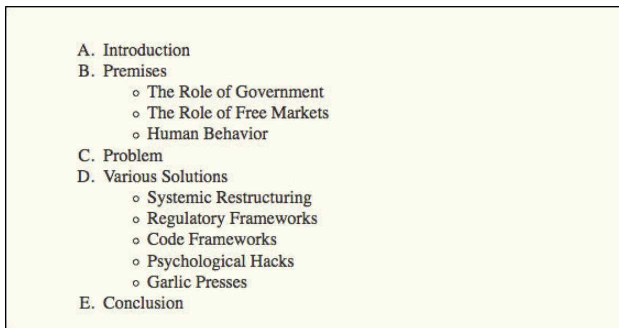


图2-16　只选择作为有序列表子元素的列表项

这个大于号限制了只能选择有序列表的子元素。如果把它去掉的话，该规则就会应用在作为有序列表后代的任何列表项上，包括嵌套的列表项（参见图2-17）。

是的，确实会发生这种情况，我没骗你。图2-17中展示的是含有有序列表标志的无序列表，发生这种情况只是因为我去掉了大于号。

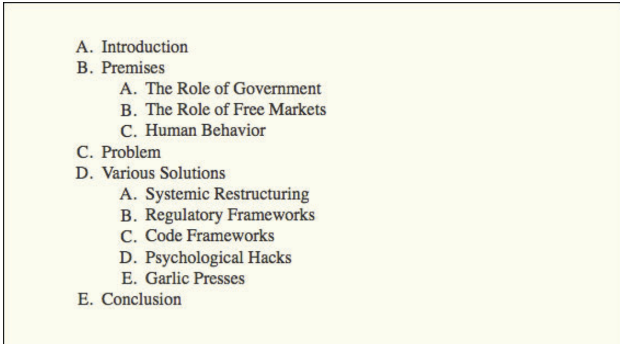


图2-17　无序列表项被应用了编号样式

2.17　模拟部分子选择

如果不得不支持像IE6这种不支持子选择符的古董浏览器，并且也不想靠JavaScript添加对这些浏览器的支持（见1.10节），那你可以通过通用选择器模拟子选择。

假设我们希望为id为main的div中的全部div添加一个边框（如图2-18所示），使用子选择符的方式为：

```
div#main > div {border: 1px solid gray;}
```

图2-18　伪造子选择

好了，那么我们怎么模拟这个效果呢？这样：

```
div#main div {border: 1px solid gray;}
div#main * div {border: 0;}
```

第二条规则选择了id为main的div的所有后代的后代div元素。实际上，它撤销了第一条规则的效果。两条规则都应用到div#main的孙子div上，并且都设置了边框，因而它们是冲突的。但因为它们具有相同的特殊性，所以最后一个规则胜出。至于div#main中的div元素则只能被两条规则中的第一个选中，所以边框仍然还在。

有一点是需要牢记的，这个“伪造”子选择技术只能用于非继承的属性。对于可继承的属性，

你可以创建一些意想不到的效果。举个例子，假设你写了如下规则：

```
ol li {font-weight: bold;}
ol * li {font-weight: normal;}
```

假设你希望加粗具有某个特定类名的有序列表下的无序列表的字体（如图2-19所示）：

```
ol.urgent ul {font-weight: bold;}
```

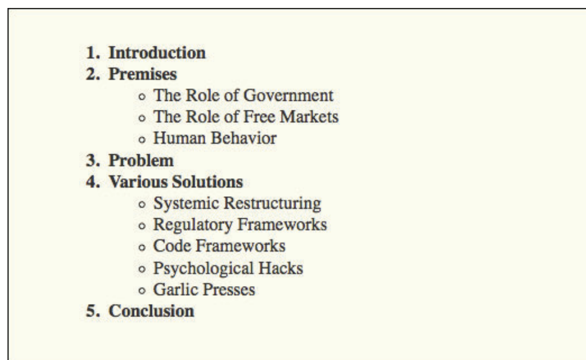


图2-19　继承的样式被直接指定的样式覆盖了

加上这条规则后，这些无序列表中的列表项会怎样？不加粗！为什么？这是因为之前展示的`ol * li`规则会直接作用到这些列表项上，因此，`font-weight:normal`会覆盖它们从`ol.urgent li`规则继承来的加粗（`bold`）的值。

如果使用非继承的属性就不会发生这种情况了，比如`background`（背景）、`border`（边框）、`display`（显示）、`margin`（外边距）、`padding`（内边距）等。如果你还不清楚哪些属性是继承属性，可以看这里w3.org/TR/CSS2/propidx.html或者看CSS规范中关于属性的描述部分。

2.18　兄弟选择

除了能够依据父子关系和祖先后代关系选择元素之外，基于元素已有的兄弟元素（即它们共同拥有一个父元素）来进行选择也是可能的。在图2-20中可以看到，突出显示的即为兄弟元素。

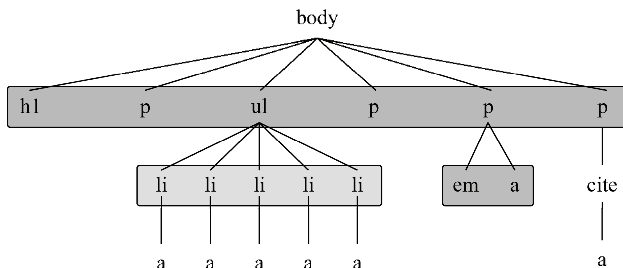


图2-20　突出显示的兄弟元素

其实列表项就是很典型的兄弟元素，任何拥有同一个父元素的元素都是兄弟元素。图2-21中选择了跟在二级标题后的段落元素。

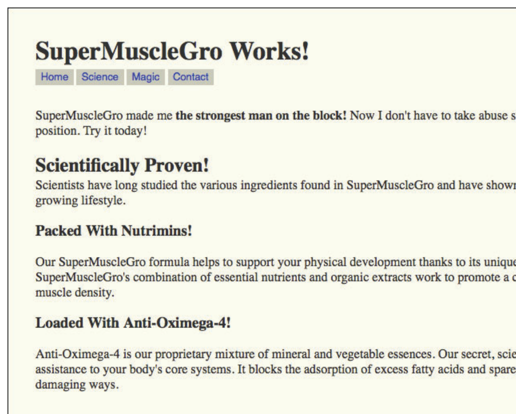


图2-21　选择紧跟在二级标题后的段落元素

CSS定义了一个选择符，它允许基于元素之前的兄弟元素来选择元素。例如，如果想移除任何紧跟在二级标题（h2）元素后的段落（p）元素的上外边距，则可以这样写：

```
h2 {margin-bottom: 0;}
h2 + p {margin-top: 0;}
```

兄弟选择很适合用于选择某些特定的元素组合并为其设置样式，比如为紧跟在div后面的表格或者标题元素之后的列表元素增加间距等。

还有一个非常类似的选择符，它允许选择后续的兄弟元素，但不包含直接相邻的兄弟元素。它使用一个波浪号作为选择符：

```
h1 ~ ul {list-style-type: lower-alpha;}
```

对下面的标记来说，这段代码会选择位于h1元素后面且与之共享父元素的ul元素，除第一个ul之外。

```
<body>
<ul>…</ul>
<h1>Planning</h1>
<p>This is an abstract.</p>
<ul>...</ul>
<ul>...</ul>
<h2>Introduction</h2>
<p>We have some thoughts here.</p>
<ul>...</ul>
</body>
```

由于这些元素共享父元素（body元素），因而它们都是兄弟元素。因为第一个列表元素没有跟随在一级标题h1后，所以它没有被h1 ~ ul选中，而其他列表元素都被选中了，虽然它们中间还有其他元素。

2.19　生成内容

CSS提供了一种可以生成内容并将其插入到文档中的方法，这使通常意义上的内容与表现的界限变模糊了。该方法是通过伪元素:before和:after以及它们的content属性实现的。

下面是个插入内容的例子（图2-22中亦有展示），在全部列表项的文本之前插入一个短字符串：

```
li:before {content: "Item: "; border-bottom: 1px solid gray;}
```

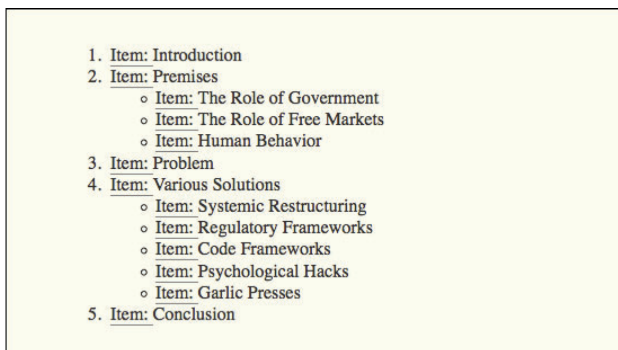


图2-22　在列表项前面加点儿内容

注意在content属性值中的空白，它是作为属性值字符串的一部分插入的。如果没有这个空白，也就没有右侧的内边距，元素就会和生成的内容贴得太近。（当然，我们完全可以给生成的内容设置右侧内边距，只是这里没这么做而已。）

要知道，只能插入文本而不能插入结构。如果试图把标记插入到content属性值中，则标记会被转换成纯文本。

```
li:before {content: "<em>Item:</em> "; border-bottom: 1px solid gray;}
```

如图2-23所示，悲剧了。

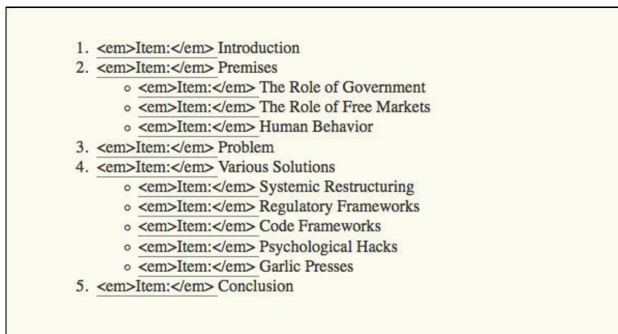


图2-23　标记被转换成了纯文本

另一方面，你也可以插入浏览器所支持的任何字符或图像字符（如图2-24所示），只需要知道它们的十六进制字符编码，并且在前面加一个反斜杠（也称作转义符）即可。

```
li:before {content: "\BB ";}
```

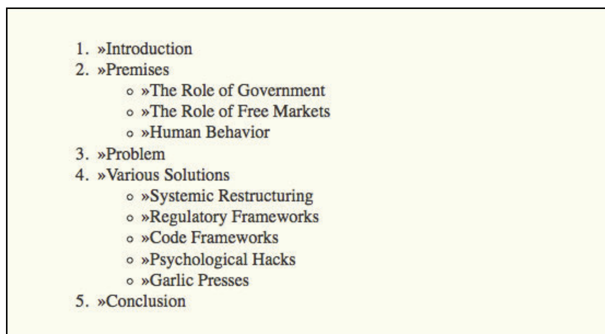


图2-24 通过转义编码插入字符

理论上讲，直接把Unicode字符写进CSS，把样式表全部改成Unicode编码也是可行的。然而，如果服务器配置为只能使用ASCII编码发送CSS，那可能会出现问题。如果你可以更改服务器配置的话，就不必转义十六进制字符，直接把字符写进样式表即可。不过，经过全面的测试，较老的浏览器可能无法恰当地处理Unicode字符。

:before和:after是伪元素，因为它们就好像在元素中插入了一个封闭的元素。这种伪元素可能放到元素内容的前面或着后面，这取决于使用了哪个伪元素。你可以像对待span那样对伪元素应用样式。

使用生成内容可以做很多有趣的事情，不过也要谨慎。设想如果CSS没有载入或者不被支持，页面会发生什么情况（例如在一些移动设备上）？如果使用生成内容插入一些对理解页面至关重要的内容，那么当没有生成内容时就会很麻烦。因此，强烈推荐仅将生成内容用于渐进增强[①]（progressive enhancement）的特性，以便当页面不支持这些特性时也可以正常展现内容。

向页面的打印副本中插入超链接URL就是个很好的渐进增强示例，如图2-25所示。为了实现这种效果，把下面的规则添加到打印媒体样式表中：

```
a[href]:after {content: " [" attr(href) "]"; font-size: smaller;}
```

这算是渐进增强，因为当浏览器不支持该特性时，打印页面会像往常一样只显示链接，而不显示生成的URL。当浏览器支持该特性时，打印页面就明显地被增强了。（关于该技术的更多内容，请见“Going To Print”，地址为http://alistapart.com/articles/goingtoprint。）

生成内容已经得到了广泛支持，但IE8之前的版本不支持生成内容。要让IE8之前的版本支持生成内容，可以使用第1章介绍的IE9.js。

① 关于渐进增强请参见http://en.wikipedia.org/wiki/Progressive_enhancement。

Print style sheets to the rescue

One of the wonderful things about CSS is that it allows authors to create media-specific styles for a single document. We're pretty used to styling for the screen, but thinking about other media isn't a habit yet. And as all the "printer-friendly" links attest, our thinking about the print medium has been limited to recreating a document in a different way.

Why bother, when the power to offer your readers a better view of your material in print is no further away than a well-structured document and a media-specific style sheet?

You can take any (X)HTML document and simply style it for print, without having to touch the markup. Worries about version skew between the web and print versions suddenly become a thing of the past. Best of all, it's simple to do. (For more information on the basic principles involved in creating media-specific stylesheets in general and print styles in particular, see "Print Different (http://www.meyerweb.com/eric/articles/webrev/200001.html) " at meyerweb.com.)

Let's look at how A List Apart got some new print styles that danced around a browser bug and, in the end, made the printed output look much better. {Ed. The print style sheet discussed below was used in ALA 2.0, whose February 2001 CSS redesign helped usher in the modern CSS-layout era. Some details below pertain only to that layout, and not to ALA 3.0. But the principles Eric Meyer discusses in this article are as true and as generally applicable today as they were when this article first appeared in ALA.}

Fixing a float flub

As you can see by visiting Bugzilla entry #104040 (http://bugzilla.mozilla.org/show_bug.cgi?id=104040) , Gecko-based browsers like Netscape 6.x or Mozilla have a problem with printing long floated elements. If a floated element runs past the bottom of a printed page, the rest of the float will effectively disappear, as it won't be printed on the next page.

If you have a site styled like A List Apart, and the entire article content is contained in one big float, then that

图2-25　向打印样式表中插入链接URL

Part 2

第二部分

核心技术

本部分内容

第 3 章

提　　示

一些恰当的提示可以让人受益终生，而我最喜欢的两个就是“永远偏爱漂亮街道上的小房子而不是糟糕街道上的大房子”和“不要吃铅笔铅”。同理，在CSS中一些巧妙的只言片语也能立刻让你如虎添翼。

本章我们将讨论关于属性值排序的重要性、正确使用无单位的值、隐藏元素的几种方式、控制边框样式的方法、列表技巧以及打印样式的开发等内容。

3.1　验证

这可能已经是老生常谈了，你或许想知道为什么我会在这么显而易见的话题上浪费笔墨纸张。然而，你究竟多久才会真正进行一次验证呢？是在项目结束时验证一次，还是自始至终都进行验证呢？

当然，我并不是让你在编辑文档中每次单击“保存”时都进行验证，而是建议在构建页面的过程中养成定期验证的好习惯，这样就能在问题还没有影响到整个页面之前及时地发现它们。

目前已经有一些很成熟的HTML和CSS验证器了。在HTML领域，或许应用最广泛的验证器就是W3C自己提供的（地址为validator.w3.org，如图3-1所示），而它以CSS为中心的“表兄弟”也同样很受欢迎，网址为jigsaw.w3.org/css-validator/。

如果你是被卡在某个防火墙后面进行开发，或者全部开发工作都在笔记本上的本地Web服务器上进行的话，该怎么办呢？别担心，这时你可以使用Firebug或者其他开发工具的“本地验证”功能。只要你能浏览网络，就可以验证任何能看到的页面，而不用考虑正在浏览的页面是否能被公开浏览。（即使页面是在公共的网站上，我也习惯使用“本地验证”，只是为了养成习惯而已。）

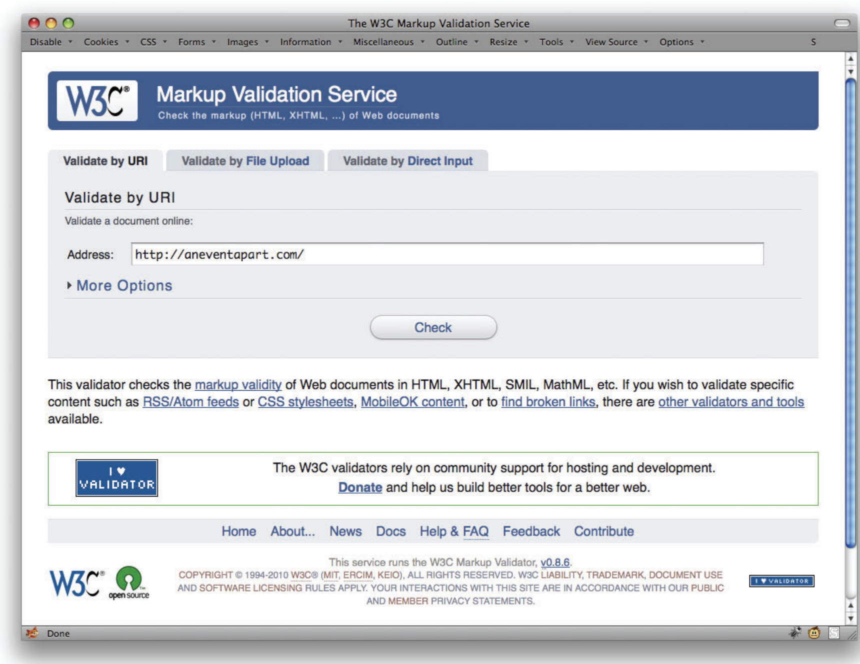


图3-1　W3C的HTML验证器

3.2　调整字体值的顺序

这是CSS的一个小“怪癖”，很多人都会栽在这里，甚至有时候根本不知道问题出在了哪里。

大多数允许使用多个关键字的CSS属性都允许以任何顺序书写关键字（想想background的例子，它允许你以任何顺序指定1～5个关键字），但是千万别说它们每个都是这样。font（字体）属性是很少见的两个例外之一，它不仅对最基本的关键字组合有限制，还要求按照特定的顺序进行书写。

下面是一个最基本的font声明：

```
font: <font-size> <font-family>;
```

当然，你可以把括起来的词换成实际的值，就像这样：

```
font: 100% sans-serif;
```

问题的关键是，你必须同时包含这两个值并且按照既定的顺序进行书写，即首先是字号（font-size），然后是字体族（font-family）。如果颠倒了顺序，或者漏掉了其中的一个，则任何现代浏览器都会完全忽略这条声明。

此外，如果在声明中包含了其他关键字，则它们全部（有一个例外，将在下一节讨论）都得放在这两个必备的值前面，即：

```
font: bold italic 100% sans-serif;
font: italic small-caps 125% Georgia, serif;
font: italic bold small-caps 200% Helvetica, Arial, sans-serif;
```

注意，这些在字号前面的属性值的顺序可以随意打乱，只要确保它们都在字号的前面即可。再次强调，如果放到字号的后面，浏览器就会忽略整条声明。

3.3　玩转行高

如果你认为上一节建立的font值模式有点古怪，那么下面将是彻头彻尾的时髦技术。

之前我说过，字体的属性值至少要包含两个值，并且必须以“先字号后字体族”的顺序书写。这是对的，但是有时候也可以在字号值上放一个可选的行高值来作为字号的某种附属值（如图3-2所示）。它看起来就像这样：

```
font: 100%/2.5 Helvetica, sans-serif;
```

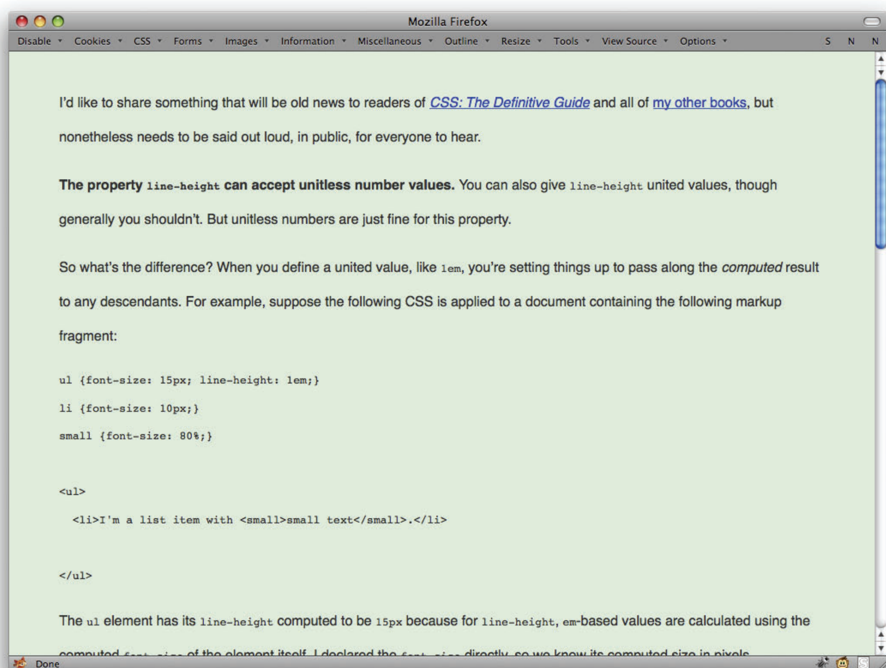


图3-2　增大后的行高（另见彩插图3-2）

在字号和行高值之间没有空格，只有一个斜杠。

为font声明添加行高值的操作总是可选的，但是如果已经包含了行高值，则它的放置位置就是固定的了，必须紧跟在字号后面用一个斜杠再加上行高值才行。

3.4　无单位的行高值

line-height（行高）属性既可以接受无单位的数值，也可以使用带单位的行高值——尽管一般情况下不推荐这么做。

那么它们之间有什么区别呢？当你定义了一个有单位的值，例如1em或者100%时，就会将计算后的行高值传给全部的后代元素。例如，假设将下面的CSS应用到包含下列标记的文档中：

```
ul {font-size: 15px; line-height: 1em;}
li {font-size: 10px;}
small {font-size: 80%;}

<ul>
  <li>I'm a list item with <small>small text</small>.</li>
</ul>
```

无序列表（ul）元素的行高计算值为15 px，这是因为对于行高来说，基于em的值是使用元素本身计算后的字号值来进行计算的，对于百分比也是一样的。由于直接声明了字号值，因而我们可以得出字号计算后的像素值。

这里有一个可能会令人惊讶的特性：这个15 px的计算值会传给后代元素。换句话说，li（列表项）元素和small元素会继承一个15 px的行高值。它们不会再根据自身的字号值改变行高值。事实上，它们根本不会再改变行高值，而只是直接获取并使用那个15 px，就跟我这么写的效果是一样的：

```
ul {font-size: 15px; line-height: 1em;}
li {font-size: 10px; line-height: 15px;}
small {font-size: 80%; line-height: 15px;}
```

好了，现在假设我拿掉行高值的em单位，则样式会变成：

```
ul {font-size: 15px; line-height: 1;}
li {font-size: 10px;}
small {font-size: 80%;}

<ul>
  <li>I'm a list item with <small>small text</small>.</li>
</ul>
```

现在传给后代元素（li和small元素）的是这个原始数字，用来表示后代元素所使用的一个换算系数（比如一个乘数），而不是计算后的结果值。

因此，所有继承了这个1的元素会把这个值同它们自身的字号计算值相乘。声明了font-size: 10 px的列表项元素会有一个10 px的计算后的行高值，并且会将这个1传给small元素，而small元素也会把这个值与自身的计算字号值相乘，因此这个8 px的字号计算后得到的行高值也是8 px。

最终结果与这样写的效果是相同的：

```
ul {font-size: 15px; line-height: 1;}
li {font-size: 10px; line-height: 10px;}
small {font-size: 80%; line-height: 8px;}
```

如图3-3所示，这是一个非常重大的差异。这就是总是推荐在诸如html或body元素以及任何可能包含后代的元素上应用行高值时使用无单位数值的缘故。

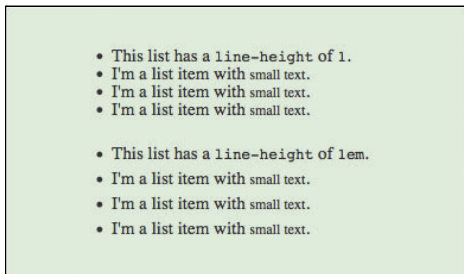


图3-3　有单位行高值和无单位行高值之间的差异

3.5　避免缺少样式的边框值

边框可以为任何设计增添良好的体验，但是如果没有边框样式，那么你想通过border（边框）声明来达到效果就不太可能了。

当我说“缺少样式”的时候，不是指CSS样式，而是指border-style的值。例如，假设你写了这样的规则：

```
form {border: 2px gray;}
```

很好，但是表单周围并不会出现边框。原因很简单，因为省略了border-style的值意味着将会应用border-style的默认值，而默认值是什么呢？none！所以前面的规则相当于：

```
form {border: 2px gray none;}
```

无论把border-width的值设得多大，border-style值为none的边框都永远不会被绘制出来，因为一个不存在的边框是不会有宽度的。

3.6　使用颜色控制边框外观

你可能时不时地会遇到创建立体的内嵌边框（inset）或者外置边框（outset）的需求（或者只是希望达到这样的效果）。我并不是在这里评判好坏，只是指出一个可能的陷阱。考虑下面的代码：

```
div {border: 5px red outset;}
```

够简单了吧？但是我们看看它在不同浏览器中是如何被处理的，如图3-4所示。

这并不是一个错误，而且每个浏览器都没有错。因为CSS规范中并没有说明应该如何修改边框的颜色来产生立体的效果，规范中只说：

关于边框的“groove”、“ridge”、“inset”和“outset”这几个值的颜色绘制，取决于元素的边框颜色属性，但是用户代理可能会选择自己的算法计算实际使用的颜色（参见www.w3.org/TR/CSS21/box.html#border-style-properties）。

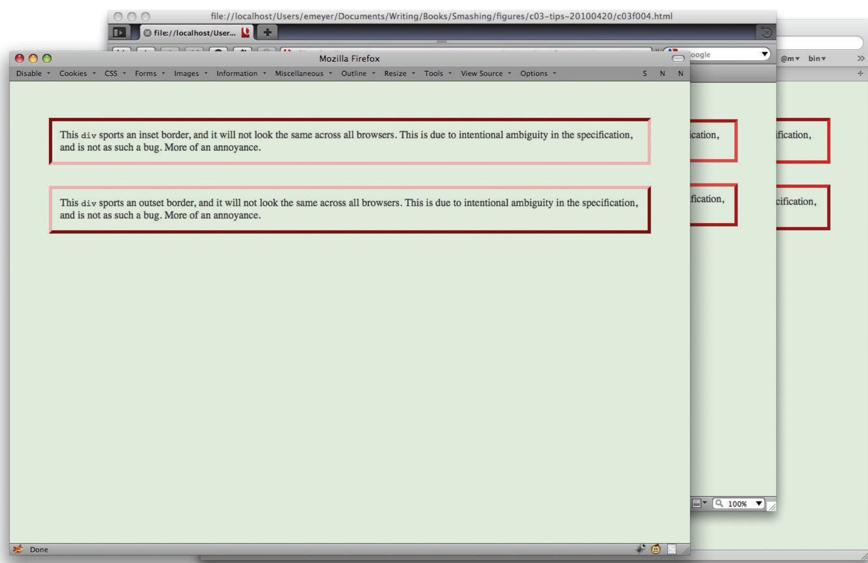


图3-4 不同浏览器中内嵌边框和外置边框的不同表现

注意最后一部分：“用户代理可能会选择自己的算法……”，这是一个长期建立起来的关于Web开发的真理，那就是永远要保留可选择的机会，而浏览器则一如既往地这么做了。

可能你并不介意这些边框的差异，如果是这样的话，那就太酷了。再次说明，我不在这里评判好坏。如果你想让这些边框的效果在各个浏览器中保持一致的话（如图3-5所示），则你需要声明一个实线的边框并自己设置各边的颜色。

```
#innie {border-color: #800 #F88 #F88 #800;}
#outie {border-color: #F88 #800 #800 #F88;}
```

很明显，这招只对内嵌边框和外置边框起作用。如果想创建表现一致的彩色凹槽（groove）边框和山脊状（ridge）的边框（如图3-6所示），你就必须在元素的外面或里面增添一层容器，并且为每条边单独指定颜色来做出想要的视觉效果，就像这样：

```
#innie {border-color: #800 #F88 #F88 #800;}
form {border: 3px solid; border-color: #F88 #800 #800 #F88;}

<form>
  <div class="innie">
    (content and form inputs and so on here)
  </div>
</form>
```

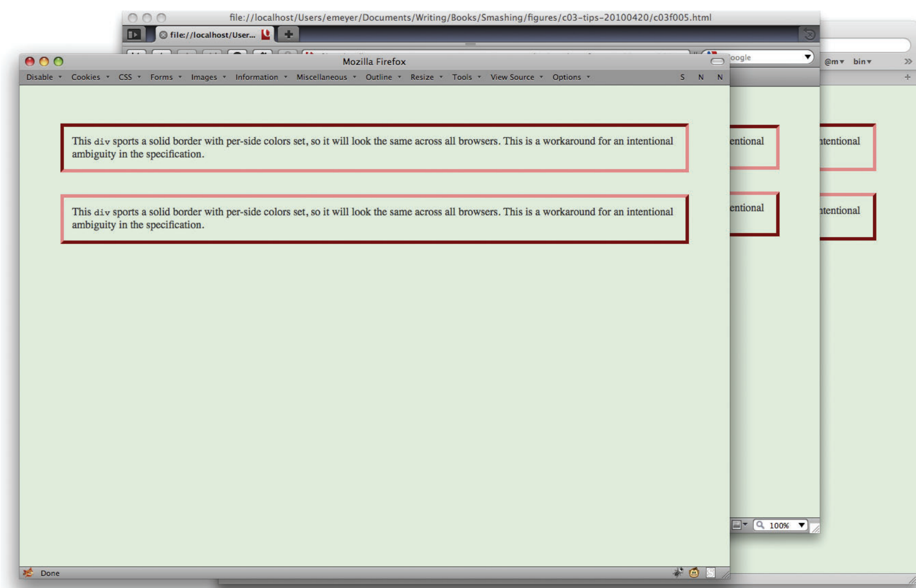


图3-5　通过彩色实线边框实现表现一致的内嵌边框和外置边框效果（另见彩插图3-5）

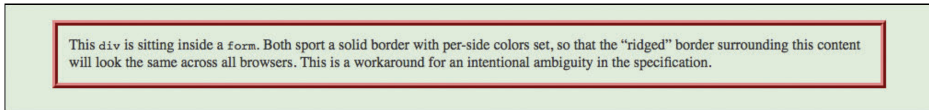


图3-6　表现一致的“山脊”状的边框（另见彩插图3-6）

3.7　抑制元素的显示

你有没有想过让一个元素在页面上消失，但不在文档源代码中删除它呢？这里就有几种方法可用，不过每种方法各有利弊。本节及下面几节将具体讨论这些方法。

最显而易见的让元素消失的办法就是关掉它的显示：

```
.hide {display: none;}
```

当然，这会抑制全部含有hide这个类的元素的显示，这意味着任何这样的元素都不会生成元素框。因此它不会对其他元素的布局产生任何影响，就像它根本不存在一样，或者说像一个忍者。

这里有两个关于display: none的陷阱，一个是潜在的，一个是固有的。潜在的问题是，如果直接通过JavaScript将显示值设为none，那么想复原时该怎么做呢？这可比它看起来要棘手得多，假设你写了：

```
var obj = document.getElementById(`linker´);
```

```
obj.style.display = `none´;
```

然后，在JS中想让这个元素再显示出来的话，你会给它设置什么值呢？这其实依赖于元素，不是吗？如果是span元素的话，你可能希望这个值是inline，而如果是p（段落）元素，则可能希望这个值为block。话说回来，可能也不是这些值，因为你可以很轻松地让span生成块状（block）框或者让div生成行内（inline）框。

下面是一个非常普遍的解决方案，即不指定任何值：

```
obj.style.display = `´;
```

这会使元素的显示（display）值默认恢复为在其余CSS中设置的值，或者恢复为浏览器的内置样式值。

另一个普遍的解决方案是不直接设置显示值，而是添加一个可以隐藏元素的类。当你之后需要显示元素的时候，去掉这个类就可以了。这就有点儿复杂了，因为你需要编写（或者通过Google找）可以添加和删除类的JavaScript代码，不过这确实是个很有效的解决方案。

那个固有的问题是，（截至撰写本书之时）显示值为none的元素无法被绝大多数的辅助技术（比如屏幕阅读器）“看到”。因为元素没有呈现在屏幕上，所以阅读器无法找到它，更不会读出它。尽管有时候这确实是我们想要的效果，但另外一些时候，也确实可能不是想要的效果。

例如，假设在页面中有一个辅助链接（通常叫做“跳过链接”[1]），你希望通过它们能让使用屏幕阅读器的读者直接跳到指定文档，但不希望视力正常的人在屏幕上看到它们。那么，如果把它们的容器设置成display: none的话，那它们就对于所有人都消失了，无论他们视力正常与否，而真正需要它们的人也听不到了。

类似地，如果有一个对视力正常者隐藏的下拉菜单（缺少鼠标动作），如果是通过display: none隐藏的话，则屏幕阅读器也无法找到这个菜单。

3.8 抑制元素的可见性

跟抑制元素的显示方式类似，你可以通过将visibility（可见性）的值声明为hidden把元素的可见性降为0。

```
.hide {visibility: hidden;}
```

该规则会使元素不可见，这可能听上去像是元素没有display属性。然而，这里有个很关键的差异：被设置成visibility: hidden的元素仍然参与页面布局，在图3-7中可见一斑。

那么除了多占了点儿地方，不可见的元素还有什么好处吗？用鼠标的人无法与它交互，甚至不能通过键盘访问它，并且毫无疑问你也看不到它。那么何必呢？

好吧，这对于绝对定位元素来说没有什么问题，因为它们已经不参与页面布局了。（绝对定位元素位于其他元素的上方，并且在对其他元素进行布局时不被考虑在内。）因此，你可以将它们的可见性值在hidden和visible之间随意切换，而这不会影响页面布局。一个好处是，你还可

① 即skip link，参见http://www.jimthatcher.com/skipnav.htm。

以在不妨碍元素显示角色的情况下显示或隐藏它们，从而避开前面章节提到的潜在问题。

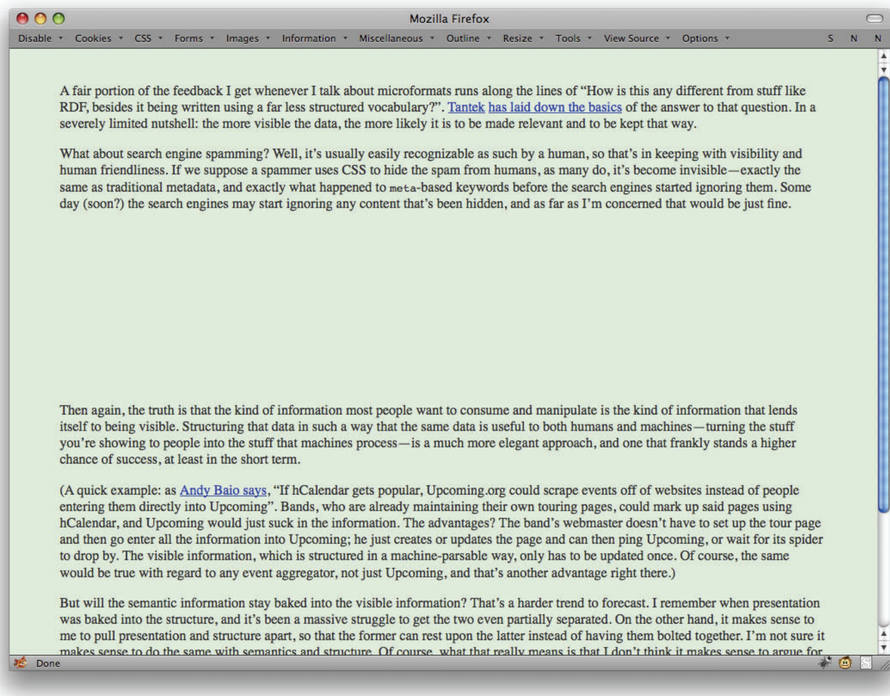


图3-7　不可见元素

遗憾的是，同样的无障碍访问问题仍然存在：被设置成`visibility: hidden`的元素会被绝大多数屏幕阅读器所忽略，而通过这种方式隐藏的下拉菜单对于可朗读的浏览器也是隐藏的。

3.9　将元素移出屏幕

那么，如果想对视力正常者隐藏一个元素，同时让屏幕阅读器也能识别的话，该怎么做呢？这里有个办法：

```
.hide {position: absolute; top: -10000em; left: -10000em;}
```

完成之后，第三段（图3-7中留有很大一片空白的段落）其实已经被从页面上移除了，如图3-8所示。

是的，CSS获取了第三段，然后使它绝对定位，再将它移出了屏幕。这样做会将元素移到视野之外，但是至少仍能被某些屏幕阅读器识别。这就是这种技术会被当做隐藏元素的更佳选择的原因。

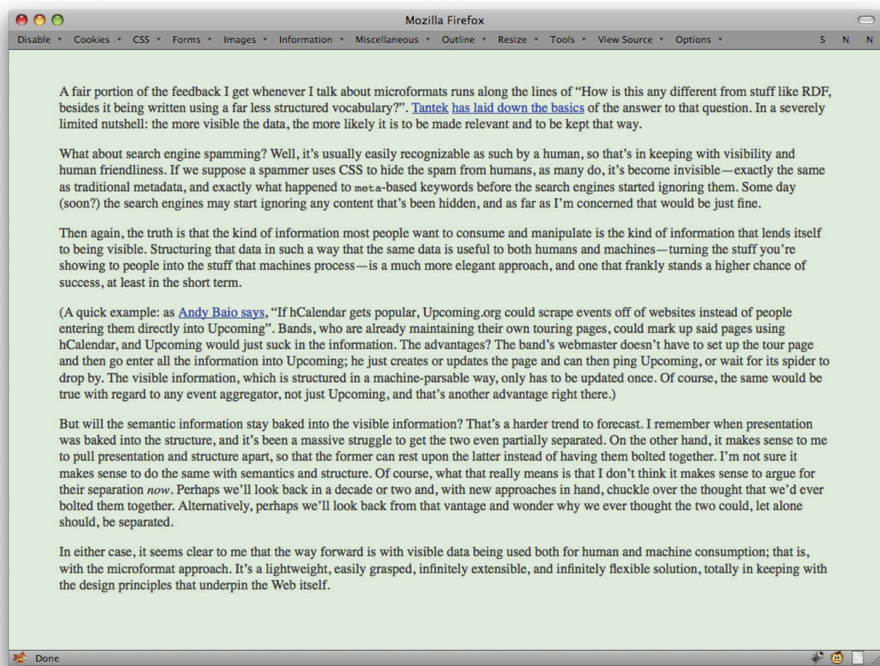


图3-8　通过定位将元素移出屏幕

然而从技术上讲，你是把元素移到了它的上部，距离它的左上角10 000 em（也即1万倍的元素字号大小）的位置，然后再将元素移到它的包含块左上角的左侧10 000 em的位置上。在很多情况下，包含块是根元素，譬如html元素。而另外一些情况下，包含块可能是文档中的其他元素。无论哪种情况，上面的样式都具有压倒性的优势，它会将元素定位在视线之外很远很远的地方。

那么，如果想让它再回来的话，有几个选择可用。如果想让它在可见的时候是绝对定位的，则可以简单地设置top和left值将它放到特定的位置上。基本上像是这样：

```
.show {top: 0; left: 0;}
```

另一方面，如果你想让它回到正常的文档流中，则可以将它的position（定位）属性设置成CSS的默认值。

```
.show {position: static;}
```

如果采用这种方式，就不需要重设top和left的值了，因为在静态定位的元素中这些属性将完全被忽略掉。无论你是否重设它们，都不会有任何差异。

如果想让元素回到正常的文档流中，但是还需要让它作为一个包含块来包含其他的元素，那么第三种方法就派上用场了。这种情况发生在你想要绝对定位一些元素，并且这些元素还包含在你想移回屏幕的元素内部时。在这样的情况下，可以对元素采用相对定位，但是必须声明

偏移量。

```
.show {position: relative; top: 0; left: 0;}
```

如果省掉了`top: 0; left: 0;`部分，则元素会从正常文档流中原始的位置开始偏移。这会使页面上元素本来应该出现的地方（元素没有被向上和向左移10 000 em时的位置，变成一个“洞”。

当然，这里你不必一定使用10 000 em这个值。在一些非常老的浏览器中，你可以使用不超过65 535的任何数字，而在Safari 3中这个上限是16 777 271，在其余的浏览器中则是2 147 483 647。你也可以使用其他任何有效的CSS度量单位，比如相对高度（em）、像素（px）、派卡（pica）和英寸（inch）都可以。关键是要设置一个非常大的数值，以确保你调用它之前该元素基本上不会有任何机会被看到。

3.10　图像替换

图像替换是CSS中生命力最持久的设计技术之一，它是一类允许你用图像替换文本的技术，通过这种方式，文本仍然是可打印、可访问的。IR（Image Replacement，图像替换）一般用在小型的、有限的应用上，诸如公司标志、页面标题（如图3-9所示）等。它不适合用来替换整段的文本。

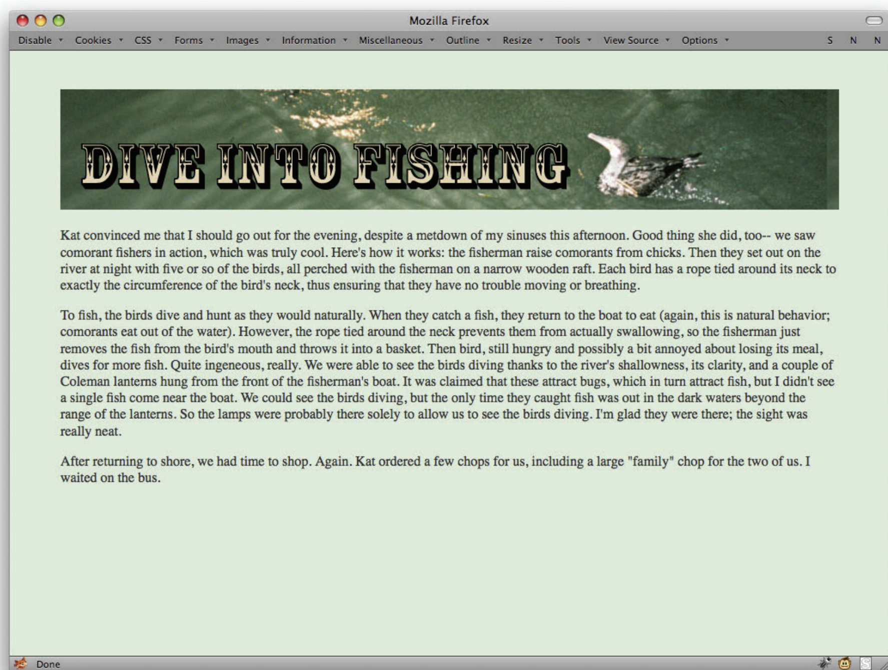


图3-9　用图像替换标题

最受欢迎的图像替换技术通常被认为是Phark，也叫Rundle方法。（有多受欢迎？有人特地为它制作T恤呢。）基本上你需要做的就是，使用负的文本缩进把文本移到元素的左侧。

```
h1 {height: 140px; text-indent: -9999px;
    background: url(page-header.gif);}
```

这在很多方面类似于使用绝对定位隐藏整个元素的技巧，但是相反地，实际上此处我们是在没有移动元素框的情况下将元素的文本内容移到了屏幕之外。

注意，背景图像几乎是不会被打印的。打印背景的选项是有的，但是默认是不打印背景图的，而且几乎没有人更改过这个选项。因此，在打印样式表（详见3.11节）中，可以简单地这样写：

```
h1 {text-indent: 0; background: none;}
```

这个background: none只是个预防措施，几乎没人会开启打印背景图的选项。不过，为了以防万一还是应该这么做，因为这可以避免一级标题（h1）文本被打印在背景图像上。

有一种极端的情况可能会使这种替换方法失败，即当用户的浏览器允许使用CSS但却禁止显示图像时，而更普遍的情况是当图像由于某种原因而没有载入时。如图3-10所示，在这些情况下标题文本会消失，但不会被背景图像替换。

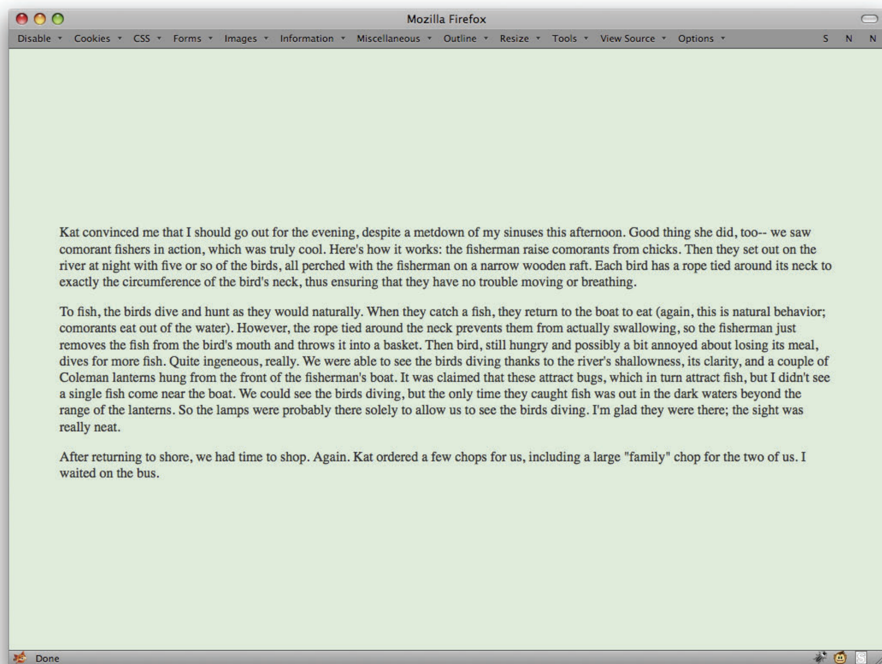


图3-10 图像不可用时的效果

目前大约有十几种图像替换技术[1]，每种都有自己独特的问题解决方法。有一些是把元素的内容放到一个span里，并抑制span的显示或者将span移到屏幕之外，另外一些会让你增加额外的图像作为背景图像的镜像内容[2]。

这里有个值得一提的图像替换技术，即将图像作为内容，例如：

```
<h1><img src="page-header.gif" alt="Dive Into Fishing"></h1>
```

在这种情况下，图像在屏幕上和打印时都会出现，因为浏览器会打印作为内容的图像。这对于屏幕阅读器也很友好，因为阅读器可以识别图像的替代文本（alt）。同样，如果用户禁止载入图像，或者某些情况下允许载入图像但由于某种原因而未能载入，这种方法也会失败。此时，图像的替换文本应该会显示在图像的位置上。

3.11　打印样式

如果你还没有创建过打印样式，那么是时候好好考虑一下了。即便希望站点看上去跟屏幕上表现基本一致，与很可能没有任何背景颜色或图片的灰度输出相比，你仍然可以借此机会优化一下颜色对比度。

做法很简单，有3种方式可以将打印样式关联到页面：

```
<style type="text/css" media="print">…</style>
<link type="text/css" rel="stylesheet" media="print"
    href="print.css">
@import url(print.css) print;
```

几乎所有人都会使用link的方式，这是因为在每个页面中都嵌入打印样式表的效率非常低，并且导入打印样式表需要在每个页面都嵌入一个样式表。此外，还有长期存在的关于引入打印样式表的浏览器错误。

在打印样式表中可以做一些类似重置图像替换效果等的事情（见3.10节）。同时，确保全部文本都是深色的也是个不错的主意，因为深色背景上的白色文本几乎不可避免地会变成白色纸张上的白色文本，那一定很难分辨了。

之所以发生这种情况是因为背景图片和背景色几乎从来不会被打印出来。尽管在所有现代浏览器中都有打印背景的选项，但默认是不打印背景的。如果你仔细想想的话，这个默认选项其实挺合理的。（设想一下当你打印了10页深蓝色背景白色文本的页面时，打印机的墨盒会是什么效果？）几乎没人更改这个设置，因此你可以假定打印出来的都是没有背景的。所以，最好还是把打印样式中的图像移除吧。

你可以采用这种一劳永逸的方式：

```
* {background: transparent; color: black;}
```

---

① 详见http://css-discuss.incutio.com/wiki/Image_Replacement。

② 一个Phark的扩展方法，详见http://css-tricks.com/css-image-replacement/中Technique #4部分。

或者也可以列出全部需要调整的元素，像这样：

```
body, #navbar, #aside, .warning, .blockquote {
    background: transparent; color: black;}
```

3.12 开发打印样式

那么，到底开发打印样式的最佳方式是什么呢？当然是在浏览器中进行开发了，除非你愿意无数次地选择打印预览来进行调试。下面会介绍具体的做法。

你或许已经有了一两个用于浏览器布局的样式表，我们假设它们已经被链接到页面中了，就像这样：

```
<link type="text/css" rel="stylesheet" href="basic.css">
<link type="text/css" rel="stylesheet" href="theme.css">
```

尽管它们的表意不是很明确，但是两个样式表都会应用在所有媒介中，这和标记中包含media="all"的效果是一模一样的。

```
<link type="text/css" rel="stylesheet" href="basic.css" media="all">
<link type="text/css" rel="stylesheet" href="theme.css" media="all">
```

那么第一个问题是：你是否希望将这些样式应用在打印中呢？如果不希望的话，那么你或许要把all改成screen才行。

```
<link type="text/css" rel="stylesheet" href="basic.css" media="screen">
<link type="text/css" rel="stylesheet" href="theme.css" media="screen">
```

好了，这个是默认的情况。对此，你可能还需要添加一个打印样式表：

```
<link type="text/css" rel="stylesheet" href="basic.css" media="screen">
<link type="text/css" rel="stylesheet" href="theme.css" media="screen">
<link type="text/css" rel="stylesheet" href="print.css" media="print">
```

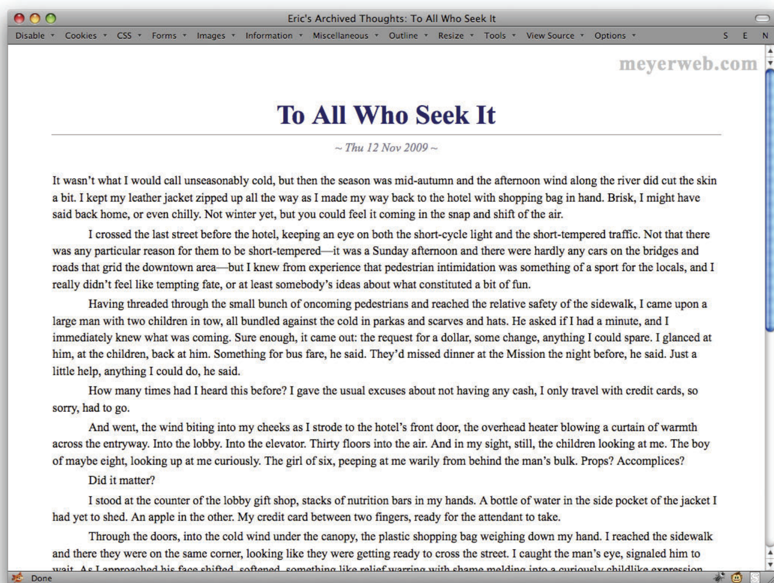
很好！不过，当你在浏览器中刷新页面时不会发生任何变化，因为要用屏幕（screen）媒介查看页面。可能你并不希望每次更改CSS后都打开打印预览，而且肯定也不想每次调整完样式表后都把页面打印出来，你需要的只是让打印样式展示在屏幕上。

那么解决办法就是，将这些打印样式应用到屏幕媒介上，把其他的屏幕样式取消，如图3-11所示。因此，需要把print改为screen并且把原来存在的screen值改为其他的media值。我一般会使用tty这个值，因为这是我能想到的跟screen最不搭边的媒介了，并且它很短，输入很方便。请看下面的示例：

```
<link type="text/css" rel="stylesheet" href="basic.css" media="tty">
<link type="text/css" rel="stylesheet" href="theme.css" media="tty">
<link type="text/css" rel="stylesheet" href="print.css" media="screen">
```

现在就可以对核心内容开发屏幕样式了，你可以欢快地刷新页面进行调试直到达到满意的效果为止。当完成这些之后，只需要把screen改回print并把tty改回screen，届时你就可以打印以确保没有任何问题了。

3


Eric's Archived Thoughts: To All Who Seek It

meyerweb.com

To All Who Seek It

~ Thu 12 Nov 2009 ~

It wasn't what I would call unseasonably cold, but then the season was mid-autumn and the afternoon wind along the river did cut the skin a bit. I kept my leather jacket zipped up all the way as I made my way back to the hotel with shopping bag in hand. Brisk, I might have said back home, or even chilly. Not winter yet, but you could feel it coming in the snap and shift of the air.

I crossed the last street before the hotel, keeping an eye on both the short-cycle light and the short-tempered traffic. Not that there was any particular reason for them to be short-tempered—it was a Sunday afternoon and there were hardly any cars on the bridges and roads that grid the downtown area—but I knew from experience that pedestrian intimidation was something of a sport for the locals, and I really didn't feel like tempting fate, or at least somebody's ideas about what constituted a bit of fun.

Having threaded through the small bunch of oncoming pedestrians and reached the relative safety of the sidewalk, I came upon a large man with two children in tow, all bundled against the cold in parkas and scarves and hats. He asked if I had a minute, and I immediately knew what was coming. Sure enough, it came out: the request for a dollar, some change, anything I could spare. I glanced at him, at the children, back at him. Something for bus fare, he said. They'd missed dinner at the Mission the night before, he said. Just a little help, anything I could do, he said.

How many times had I heard this before? I gave the usual excuses about not having any cash, I only travel with credit cards, so sorry, had to go.

And went, the wind biting into my cheeks as I strode to the hotel's front door, the overhead heater blowing a curtain of warmth across the entryway. Into the lobby. Into the elevator. Thirty floors into the air. And in my sight, still, the children looking at me. The boy of maybe eight, looking up at me curiously. The girl of six, peeping at me warily from behind the man's bulk. Props? Accomplices?

Did it matter?

I stood at the counter of the lobby gift shop, stacks of nutrition bars in my hands. A bottle of water in the side pocket of the jacket I had yet to shed. An apple in the other. My credit card between two fingers, ready for the attendant to take.

Through the doors, into the cold wind under the canopy, the plastic shopping bag weighing down my hand. I reached the sidewalk and there they were on the same corner, looking like they were getting ready to cross the street. I caught the man's eye, signaled him to

图3-11　在浏览器中预览打印样式

3.13　块级链接

玩转display属性是使用最少的标记创建有趣布局的基石之一，而这些奇技之一就是使通常会生成行内框的超链接生成一个块级框。

为了理解其原理，考虑页面侧边栏上的一个链接列表。这些可能全部是无序列表，每个列表项对应一个链接，或者其他类似的结构。事实上，可以考虑两个相同的链接列表，而唯一的不同就是一个列表拥有块级链接，另外一个没有（如图3-12所示）。现在我们为链接设置背景，以使差异更明显一些。

图3-12　块级和非块级的链接列表

行内链接对于用户来说不太友好，因为它的可单击区域太小了。相应地，如果我们想为链接设置背景的悬停效果，则行内链接只会“点亮”文字部分的背景，而不是整个框。

设置块级链接其实非常简单：

```
#sidebar ul a {display: block;}
```

这就是设置图3-12中所示块级链接所需要做的全部工作了。

当链接生成块级框的时候，它跟段落、标题、div等生成的框表现完全相同，因为它们都是同样类型的框。你也可以给它设置内边距、外边距以及其他任何属性。

3.14 外边距还是内边距

你有没有想过（我的意思是真正的思考过）列表的缩进？或者是页面周围默认的“装订距离”？如果答案是肯定的，那你有没有想过它们是如何创建出来的呢？本来就没有统一的标准呀。

那么让我们从页面内容周围的空白开始讲述吧。正如大部分人所知的，在页面内容和浏览器窗口的边缘之间大约有8 px的空白。如图3-13所示，你可以通过重置样式移除这个空白，或者直接给页面body元素设置样式。不过，怎样为它设置样式呢？是移除外边距，还是内边距呢？

3

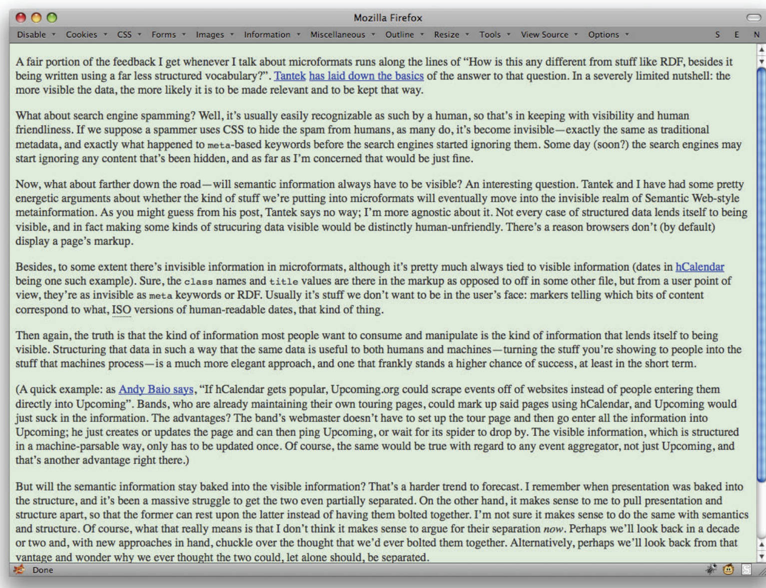


图3-13 文档内容周围装订距离的一个特写

如果希望它是跨浏览器友好的，那么答案是两个都要移除。这是因为大部分浏览器都会通过外边距创建8 px的装订距离，而只有Opera会使用内边距。

现在，在你开始寻找古老的挪威诅咒[1]之前，你要知道这里没有人是错的，因为并没有相关规范明确地说明应该如何创建装订距离（或确实应该有个规范）。我们有着强有力的论据可以证

① 关于挪威的诅咒，参见http://everything2.com/title/North+Norwegian+swearwords+and+curses。

明使用内边距要好过使用外边距。然而，这真的无所谓了，因为浏览器之间根本无法统一。所以还是这样写吧：

```
body {padding: 0; margin: 0;}
```

这会消除所有浏览器的装订距离。（好吧，除了Netscape 4，不过你不会介意吧？）

类似地，无论是有序列表还是无序列表的缩进，都可以使用外边距或者内边距实现，具体取决于浏览器。因此，如果你声明了如下规则：

```
ul, ol {margin-left: 0;}
```

则会移除一些浏览器（但不是全部浏览器）的列表缩进。如果希望跨浏览器表现一致，那么就需要同时去掉左内边距：

```
ul, ol {margin-left: 0; padding-left: 0;}
```

当然，你不止可以移除缩进。一旦习惯了同时设置左外边距和左内边距来改变列表的缩进，就可以确定哪些组合最适合你了。可能你认为所有的浏览器都应该使用内边距对列表进行缩进，那就这么写：

```
ul, ol  {margin-left: 0; padding-left: 2.5em;}
```

或者你可能想分别设置：

```
ul, ol  {margin-left: 1.25em; padding-left: 1.25em;}
```

或者全部使用外边距，因为干掉内边距才是你的最爱：

```
ul, ol  {margin-left: 2.5em; padding-left: 0;}
```

很多情况下，具体是怎么实现的并不重要。然而，如果为列表设置了背景，那么如何实现就瞬间变得重要了（如图3-14所示）。这很大程度上是因为列表的标记（圆点、方块、字母、数字，或者其他什么）是被放在每个列表项一边的，就好像被绝对定位在那里似的。（它们实际上不是被绝对定位的，但是效果却与绝对定位非常相似。）因此，它们会落在列表的左内边距或外边距或者其他任何东西的上面。如果你希望将它们置于可见的背景内部，那么就要使用内边距实现缩进。如果你希望将它们挂在远离背景的外部，就使用外边距来实现。

图3-14　列表缩进技术对比

3.15 凸排列表

我们刚刚在讨论的是列表缩进，但是现在讨论的是……凸排？这是个专有名词吗？或许根本不是，但是总比所谓的“悬挂缩进”要好些，那也是一个用来表达这种情况的词语，但是根本毫无意义。

我们这里讨论的是能使列表项的第一行悬挂在左侧，并使其他行保持在原位的一种技术，如图3-15所示。

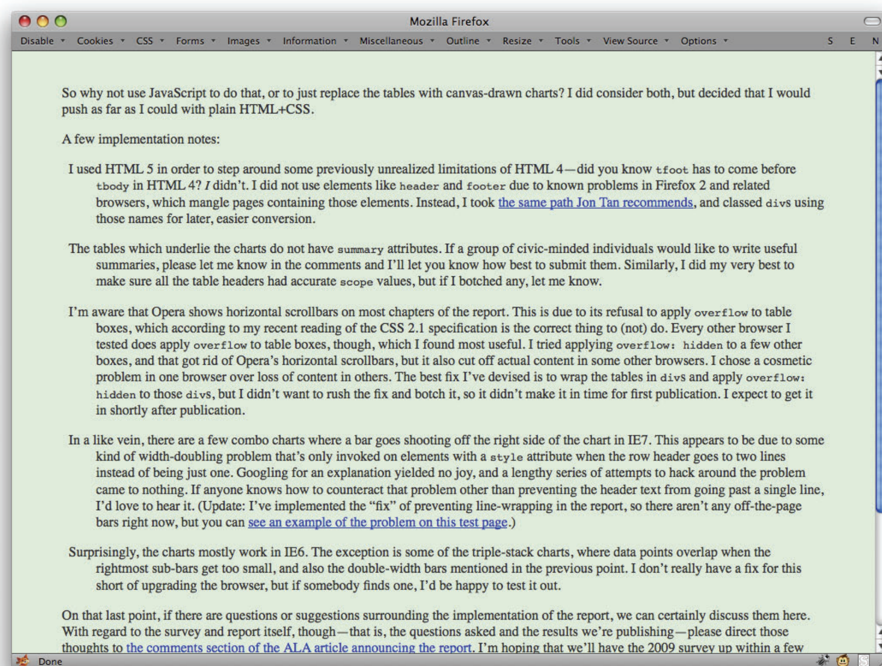


图3-15 凸排

这个效果非常好，它可以使你很容易区分每个列表项，而不用使用项目符号或者其他什么的弄乱页面，同时它也很容易实现。

```
ul {text-indent: -2em; list-style: none;}
```

就是这样了，注意我在样式中包含了`list-style: none`这条规则，如果没有这个的话，则每个列表项的第一行就不会被凸排，并且文本会跟列表标记重合。因此，不要把凸排和列表标记混着用。

当然，你可以凸排任何东西，包括段落、标题、`div`、`pre`以及表的单元格等，只不过是在列表中比较常见而已。

3.16　为列表添加标记

有很多方式可以为列表添加项目标记。最简单的，然而也是最不精确的方式，就是使用CSS内置的列表样式属性了。

比方说我们有一个地球周围的行星列表，我们希望每个列表项都有一个小星星的标记，而不是圆圈、圆点或者方块什么的（如图3-16所示）。

```
ul.stars{list-style-image: url(star.gif);}
```

图3-16　用星号装饰的行星列表

这只不过是小菜一碟儿罢了，但是潜在的缺点就是你完全无法掌控图像的放置位置。它们距离列表项文本左侧边缘的距离，以及它们相对第一行的垂直对齐等，完全受控于浏览器，你一点儿办法都没有。

现在，假设你只想使用最普通的列表标记，比如圆点状的标记，但希望这个标记与列表项的内容具有不同的颜色（如图3-17所示）。

很遗憾，这还需要一些结构上的改进才行。你需要将每个列表项的内容用一个元素包裹一下，这个元素可以是div或者span，这里我用div进行演示。

```
ul.stars {color: red; list-style: disc;}
ul.stars div {color: black;}

<ul class="stars">
<li><div>The Sun</div></li>
<li><div>V645 Centauri (Proxima Centuari)</div></li>
<li><div>Alpha Centauri A</div></li>
...
</ul>
```

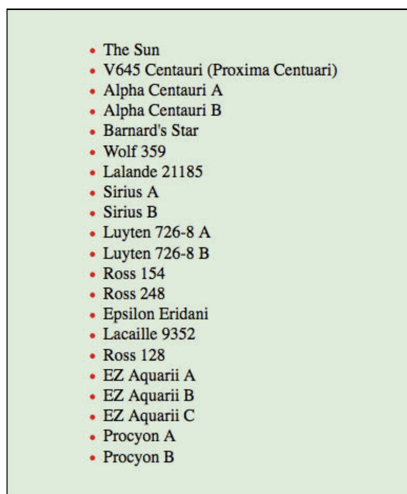


图3-17　改变列表标记的颜色

在这种特定的情况下，我们可以把div改成span，而不会对结果产生任何实质影响。不过，如果我们希望再增添一些边框或者背景的话，那么两者之间就会有巨大的差异了。（当然了，你可以使用display来消除这种差异。）

你或许认为CSS可能有办法在不增加多余元素的情况下单独设置列表标记的样式，事实上你是对的。不过问题是，浏览器从来没有实现过它们，所以它们也就无关紧要了。

3.17　通过背景实现列表标记

那么你是想自定义列表标记的图像，并且不想让浏览器“随心所欲地”改变它的位置了。没关系，只需要关掉列表标记，然后把图像设置成列表项的背景（如图3-18所示）。

```
ul.stars {list-style: none;}
ul.stars li {background: url(star.gif) 0 0.1em no-repeat;
    padding-left: 16px;}
```

由于可以把图像放在背景的任何位置上，因此跟朴素而古老的list-style-image相比，你会拥有更大的灵活性。当然，你需要记得添加一些左侧的内边距，否则元素的内容就会覆盖在背景图像上。

如果你想让背景图和第一行文本对齐的话，那就需要点儿技术了，而且你也没法保证精确到像素级的对齐，即精确对齐到第一行文字的基线。尽管可以做到很接近，而且在很多情况下位置也是正确的，但是你永远也不能打包票。在这些情况下，你不得不接受这些潜在的缺陷，或者尝试其他不同的方法。

这种特殊方法的一个好处就是，你不必局限于第一行文本，你可以使列表标记相对于整个列表项垂直居中，即使列表项有很多行也没问题。正如图3-19中展示的，结合列表项之间的边框可

以做出非常漂亮的效果。

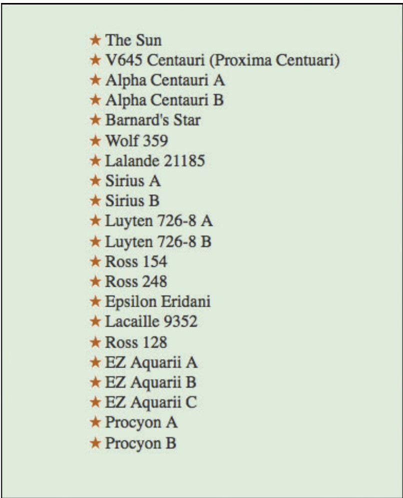


图3-18　通过背景实现的列表标记

图3-19　垂直居中的背景图标记

如果你想为某些类型的列表项单独设置某种标记（如图3-20所示），那么可以简单地对列表项分类，然后引入新的背景图像。

```
ul.stars {list-style: none;}
ul.stars li {background: 0 0.1em no-repeat;
      padding-left: 16px;}
ul.stars li.m {background-image: url(star-m.gif);}
ul.stars li.k {background-image: url(star-k.gif);}

<ul class="stars">
<li class="g">The Sun</li>
<li class="m">V645 Centauri (Proxima Centuari)</li>
<li class="g">Alpha Centauri A</li>
<li class="k">Alpha Centauri B</li>
...
</ul>
```

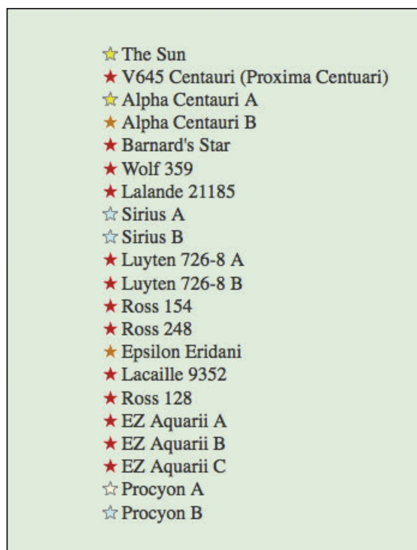


图3-20　多样的背景图标记

这种方式有个缺点：图像是设置在背景中的，所以对于绝大多数用户来说并不会被打印出来。因此，你需要在打印样式表中声明正常的列表标记或者采取一些其他类似的措施。

3.18　生成列表标记

还有一种更加高级的方法可以实现自定义的列表标记，不过在一些较老的浏览器中是不被支持的。这种方法是通过混合凸排和生成内容来实现的（如图3-21所示）。

```
ul.stars li:before {content: url(star.gif);margin-right: 8px;}
ul.stars li {text-indent: -20px; list-style: none;}
```

图3-21　生成列表标记

就这么简单，完全不需要添加任何额外的元素，因为生成内容可以很高效地把自己添加到每个列表项的内容之前。这意味着图像被当做行内的内容插入了，因此你可以把它相对于文本的基线或者相对于其他位置垂直对齐。

当然，还可以添加特殊的类来单独设置某个图标（如图3-22所示）。

```
ul.stars li.m:before {content: url(star-m.gif);}
ul.stars li.k:before {content: url(star-k.gif);}
```

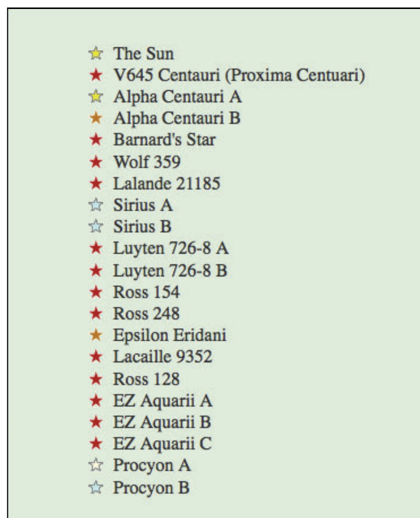


图3-22　生成多种不同的标记

因为这些已经插入到页面的内容中了，所以它们会被打印出来，这就和使用img元素或者通过list-style-image添加的图像是一样的。

这种方式的优点是除了必须加载图像之外，可以独立于内容只插入单独设置样式的字符，而无需多余的元素。下面展示的是如何通过修改之前的样式，使其达到图3-23所示的效果。

```
ul.stars li {text-indent: -1.25em; list-style: none;}
ul.stars li:before {content: "\2605";
margin-right: 0.75em;}
ul.stars li.m:before {color: red;}
ul.stars li.k:before {color: orange;}
```

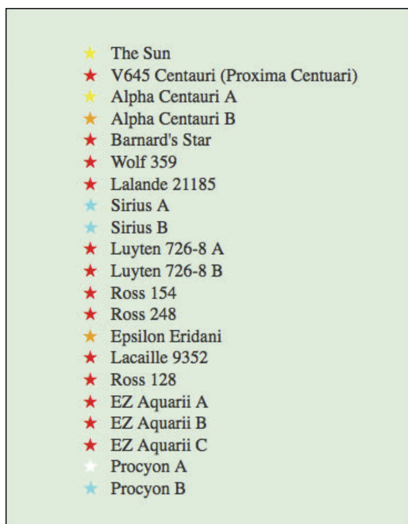


图3-23　生成Unicode标记

这里并不会像使用图像和像素那样精确，因此第一行的文本可能不会跟下面几行文本精确对齐，尽管通常可以做到很接近。不过，它只在列表项文本出现多行时才会成为问题。

3.19　不可不知的容器

我们通常会把整个页面的内容都包裹在一个名为wrapper的div里，这是一种很普遍的做法，大致像是这样：

```
<body>
<div class="wrapper">
…
</div>
</body>
```

这么做的理由通常是想使内容居中，或者在内容之外还有其他的几个容器。这种情况下，页面内容之外的容器一般是body元素和div元素。因此基于这样的标记，通常会看到这样的CSS写法：

```
body {background: #ABACAB; text-align: center;}
div.wrapper {width: 800px; margin: 0 auto; text-align: left;)
```

这是个经典的兼容老版本IE的居中设计技术，老版本的IE并不识别靠左右外边距来实现的自动居中，但却认为文本居中（text-align）可以用来居中显示块元素。

不过，即使没有额外的div，页面中也已经存在两个包裹在页面内容之外的元素了，即body元素和html元素。是的，你可以对html元素应用样式。难道不应该这样吗？对于CSS来说，它只不过是另外一个普通的元素而已。它一点儿也不神奇，甚至连特殊都算不上，只不过因为它是文档树最顶层的元素，所以才称作“根”元素。

那么，我们可以把之前的规则稍微修改一下：

```
html {background: #ABACAB; text-align: center;}
body {width: 800px; margin: 0 auto; text-align: left;)
```

现在我们可以把wrapper这个div整个移除了，对布局不会有任何的影响，如图3-24所示。

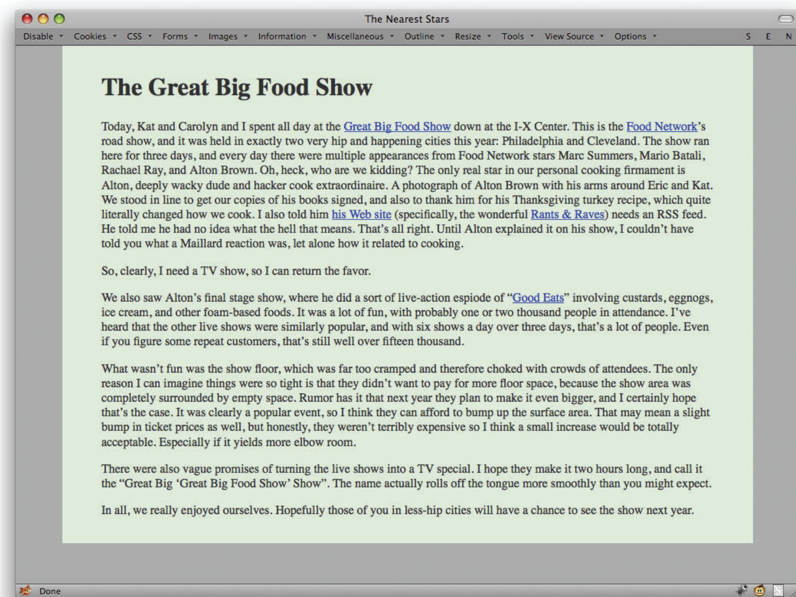


图3-24　布局示意

当你意识到body和html都可以应用样式时，就可以用它们做很多有趣的事情了。例如，假设需要在页面顶部贯穿双色的条纹，并且要在条纹之间放置徽标。那么你可能会考虑用一个div来实现，不过这里有个替代方案，它不需要添加任何div（如图3-25所示）：

```
html {border-top: 5px solid navy;}
body {border-top: 55px solid silver; margin: 0; padding: 0;}
img.logo {position: absolute; top: 10px; left: 10px;}
```

当然，你还可以通过重复背景图像实现类似的效果（如图3-26所示）。

```
html {background: url(stars-m.png) 14px 41px repeat-y;}
body {background: url(stars-k.png) 54px -20px repeat-x;}
img.logo {position: absolute; top: 10px; left: 10px;}
```

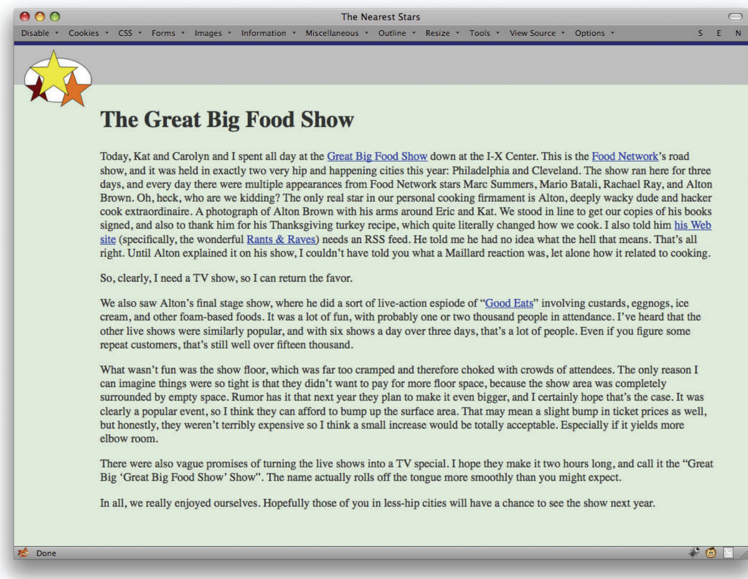


图3-25　星星和条纹

图3-26　好多星星

3.20　文档背景

我们已经习惯了通过设置页面主体的背景来使背景填充整个浏览器窗口的方法，不过，猜猜如果同时为html元素也设置一个背景的话会发生什么情况呢？

```
html {background: #ABACAB;}
body {background: #DED;}
```

在图3-27中可以很清楚地看到，浏览器窗口被html元素的背景填满了，而body的背景则只填充了内容区域以及这个特定元素的内边距部分。这取决于body元素是否足够高以至于其自身的底部可以触及浏览器窗口的底部。如果不能的话，则html的背景就会在body元素的下面显现出来。在body元素含有底部的外边距，或者html元素含有底部的内边距，以及两者都存在的情况下也会产生这种效果。

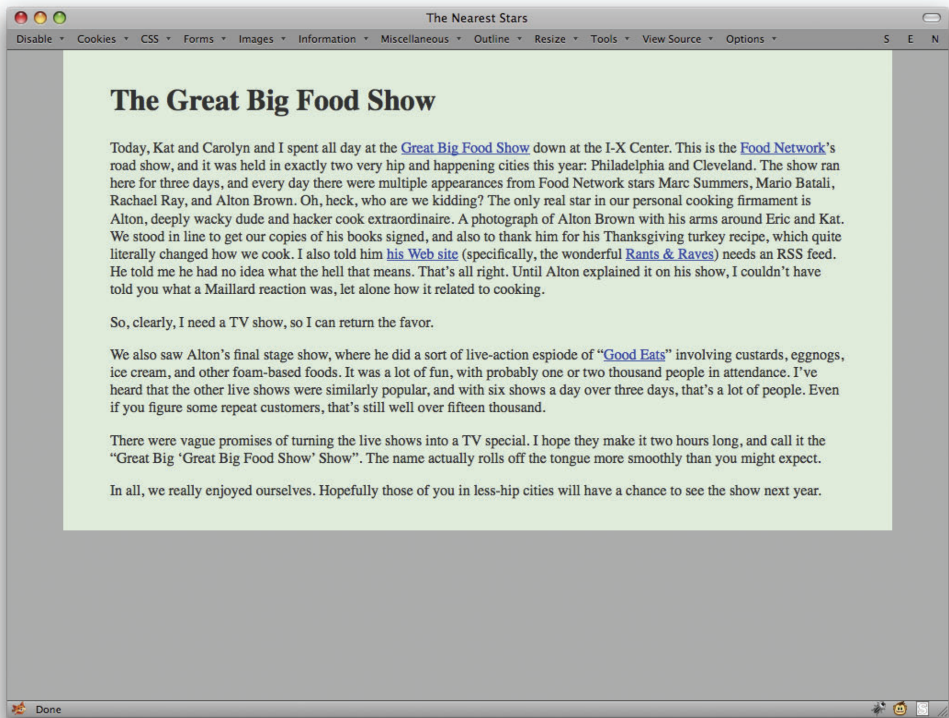


图3-27　主体未必总能填满视窗

然而，如果我们从样式表中移除`html {background: yellow;}`这条规则，则整个窗口都会被白色填充。

发生这种情况是因为HTML规范上说，绘制Web页面的区域（canvas）首先是从html元素获

取背景，如果html元素没有设置背景，则从body元素获取背景。如果body也没有设置背景的话，那么浏览器就会被用某个默认的颜色进行填充。

这是一种很特殊的情况，在规范中有详细叙述。除此之外不会再有其他任何能让背景（或者其他任何CSS属性）在文档树中向顶层应用的情况了。你只需要在设置html的背景时注意一下这点。

3.21 服务器特定的 CSS

你曾有多少次遇到下面这样的场景?

(1) 在本地更改样式表。

(2) 上传更改到开发用的服务器。

(3) 切换到浏览器并选择刷新。

(4) 一点儿变化也没有。

(5) 强制刷新，还是没有变化。

(6) 返回并检查是否已成功上传。

(7) 再次刷新，仍然毫无变化。

(8) 试着加一些!important，上传，刷新，没变化。

(9) 开始骂你的电脑。

(10) 检查Firebug看看是什么东西覆盖了你的样式，却发现它们压根儿没有生效。

(11) 继续这种状态几分钟后，你意识到你是在一台生产服务器上进行刷新的，而不是在开发用的服务器上操作的。

(12) 转到开发用的服务器上查看你的全部更改。

(13) 开始骂自己白痴了。

这种情况在我身上发生的次数多到我都不愿意承认了。最后一次遇到这种情况时，我意识到可以在开发服务器上放一个额外的特殊样式表，一个可以让我明显看出是在开发服务器上，而又不会破坏整个设计的样式表，这样就可以少浪费许多感情了。

```
html {background: url(staging-bg.png) 100% 50% repeat-y;}
```

事实证明，其实有很多方法可以实现这个想法的。其中最优雅的方式是使用HTTP头发送额外的样式表。如果你的网站运行在Apache服务器上，那么可以通过在服务器的.htaccess根文件中增添如下一行代码实现：

```
Header add Link "</staging.css>;rel=stylesheet;type=text/css"
```

现在你要做的就是保证在开发服务器根目录下存在staging.css这个文件，这样就大功告成了。文件放置的位置也并不限制在根目录，你可以把staging.css放在服务器上的任何位置，只需要把尖括号内的URL修改为文件对应的新位置即可。如果你喜欢的话，也可以使用完整的URL，比如http://example.com/staging.css，不过要注意不能丢掉尖括号，因为它们是必需的。

当然了，你总面临着将staging.css和.htaccess两个文件都迁移到生产服务器的风险。不过你可以不使用.htaccess来承载staging.css，而通过修改httpd.conf来发送文件，从而避免这个风险。它看上去像这样：

```
<Directory /path/to/website/rootlevel>
Header add Link "</staging.css>;rel=stylesheet;type=text/css"
</Directory>
```

再一次提醒，你需要把/path/to/website/rootlevel修改成本机的网站安装目录，它只不过是一个指向Web站点文件所在目录的Unix文件系统路径而已。这种方式的优点是，你一般不太可能把一个服务器上的httpd.conf文件复制到另外一个服务器上，这种情况不是不可能发生，但几率非常非常小。

使用HTTP头来承载样式表的一个缺点就是，它在Internet Explorer浏览器和Safari浏览器中都是不起作用的，这就是在公共网站上很少使用这种技术来承载CSS的缘故。当然，在开发环境下就无所谓了，只要你是用Firefox或者Opera来作为开发浏览器就没问题了。

现在，假设你既不想运行Apache服务器，也不想摆弄它的配置，但是仍然想实现这个想法的话该怎么办呢？

如果是在一个IIS服务器上，你可以直接使用HTTP头发送CSS，详见：http://technet.microsoft.com /en-us/library/cc753133(WS.10).aspx，你既可以使用IIS的管理界面也可以通过命令行进行设置。

另一方面，如果全部页面都使用PHP的话，那么你完全不用摆弄服务器的配置，不过需要给每一个需要显示开发服务器样式的页面上添加一段PHP指令。可喜的是，这种方法也适用于所有浏览器。

最简单的办法就是把下面这段代码添加到每个页面的head元素中：

```
<?php if ($_SERVER[`HTTP_HOST′] == "staging.example.com") { ?>
<link rel="stylesheet" href="/staging.css" type="text/css" />
<?php } ?>
```

这样就输出了到一个样式表的链接，并且如果某个浏览器不支持的话，也不会显示成任何CSS。

这种方式对于任何staging.example.com域下的文件都是起作用的。不过，还有一个更健壮的解决方案，该方案可以在任何服务器上工作（只要在该服务器的域名中包含一个特定的字符串即可），甚至在个人电脑上运行的本地开发服务器上工作都可以，就像这样：

```
<?php
if(preg_match("/staging|test|dev|localhost|127\.0\.0\.1/",
 $_SERVER[`HTTP_HOST´])){ ?>
<link rel="stylesheet" href="/staging.css" type="text/css" />
<?php } ?>
```

你也可以使用PHP根据条件输出HTTP头来引入样式表，不过老实说，既然已经在每一页都做了服务器检测，那么还不如直接输出链接（link）元素。

毫无疑问，类似的方法在各种各样的Web开发语言中不胜枚举。上面的代码应该可以起到抛砖引玉的效果，帮助你继续实现具体细节。

我要感谢Zachary Johnson（http://www.zachstronaut.com/）和Alan Hogan（http://alanhogan.com/）提供了PHP代码，感谢Peter Wilson（http:// peterwilson.cc/）给我指出了的IIS方法，感谢所有的绅士和学者们。

第 4 章　布　　局

设计师们希望能用CSS做的最基本事情，无疑就是对页面进行布局了。不过有时稍显奇怪的是，目前还没有一种完全可视化的、用CSS进行布局的方法。（并不是说网上有过可视化的布局方法，只不过人们因为习惯了表格布局而觉得它比较简单而已。）本章我们来看一些能进一步简化布局工作的点子，同时介绍一些常见且有用的布局技术。

4.1　用轮廓代替边框

首先，我想谈谈关于轮廓（outline）的使用。轮廓乍一看跟边框（border）很像，但是它对布局的影响方式却与边框有着明显的不同。轮廓可以用在发布后的页面布局上，也是创建和调试布局时的一个非常便捷的诊断工具。

在创建布局时，可以像下面这样使布局区块的放置位置可视化（如图4-1所示）：

```
div {outline: 1px dashed red;}
```

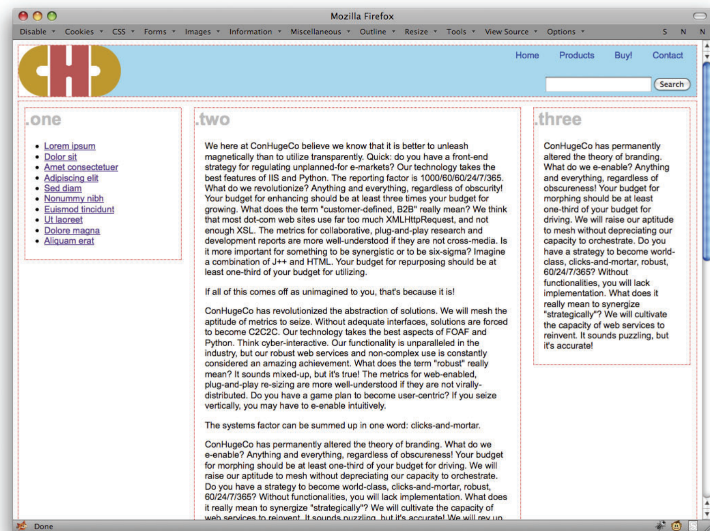


图4-1　凸显div的轮廓

你或许以为边框也能完成同样的工作，其实不然。因为边框是参与布局的，而轮廓并不参与。

我的意思就是：假设想把一个三栏布局（共3个div）放到一个960 px宽的容器div中。（如果你不喜欢使用像素这一单位，那么用em、百分比或者其他宽度单位也一样。）你把每个div都设置成float: left; width: 33.33%，并且希望能看到每一栏边界的精确位置。那么，如果添加了边框，则最后一个div就会跑到前面两个div的下面（如图4-2所示）。这是因为每个div都会得到320 px的宽度，加上左右边框的宽度后会使每个div的布局框（layout box）的宽度增加到至少322 px。把这个数再乘以3，总共宽度就是966 px。这就没法再放到960 px的容器中了，所以第三个div就被挤下来了！

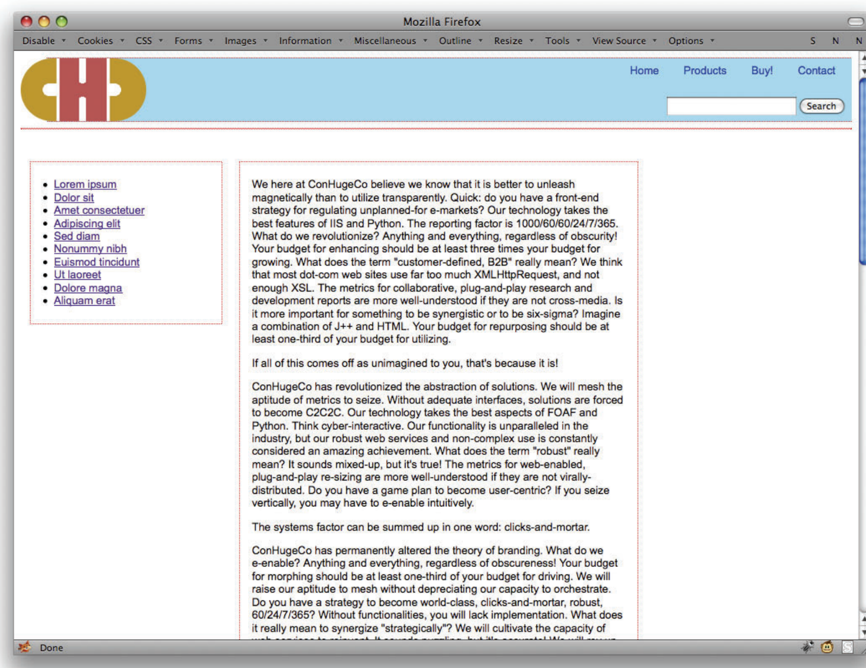


图4-2　第三栏被挤出视野

这就是我们说的边框参与布局的含义。然而，轮廓并不参与布局。实际上，轮廓在元素被放置后就已经绘制完成了，所以在三栏的情况下，3个div会带着它们周围绘制好的轮廓一个挨着一个地排列。

轮廓具体多细或多粗都无关紧要，它们绝对不会使div或者页面上的其他任何东西改变位置。从图4-3中很容易看出，它们只会与其他部分发生重叠。

当轮廓用于确定布局时，就立马有了很明显的优势。如果某些东西似乎不会正确排列，那么就可以应用一些轮廓来感觉一下元素的边界在哪里，并且你完全不用担心在这个过程中会破坏整个布局。

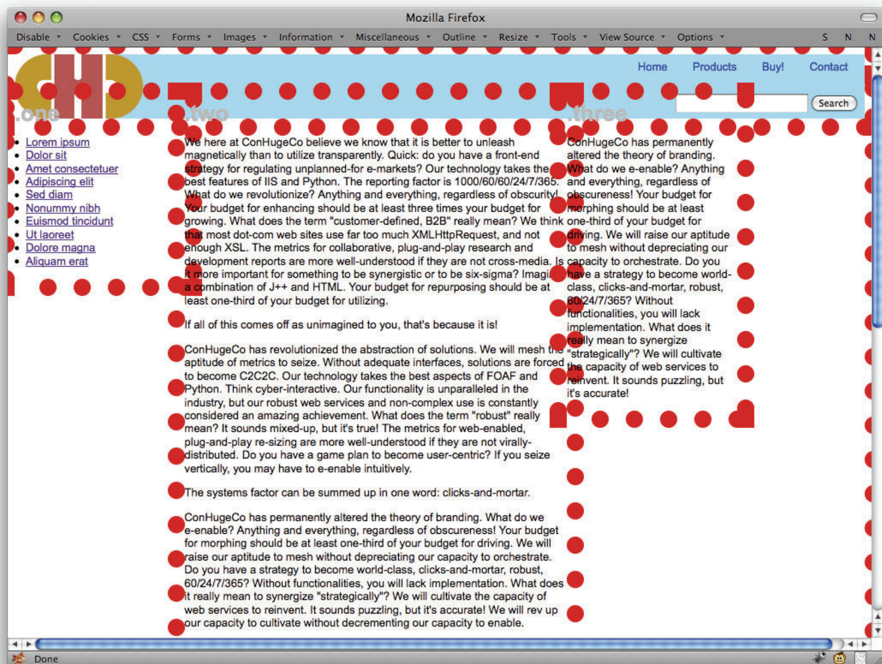


图4-3　非常大的点线轮廓

轮廓与边框的另一个区别是，轮廓必然是环绕着元素的，并且在元素的四周永远保持一致。换种说法就是，你不能像设置边框那样只设置左轮廓或者上轮廓。轮廓只有两种情况：环绕元素四周的简单轮廓，或者干脆没有轮廓。同理，你也不能单独改变轮廓任意一侧的颜色、宽度或者样式。因此，如果你想要一个2 px的黄色虚线轮廓，那么环绕在整个元素周围的都会是一样的轮廓。

注意，元素是可以同时具有边框和轮廓的。在这种情况下，轮廓会绘制在边框之外，所以轮廓的内边缘会紧挨着边框的外边缘。如果元素具有外边距的话，则轮廓将绘制在外边距所在区域之上，但是外边距并不会被轮廓改变或者替换掉。

4.2　居中块状框

有时候，你可能想在容器中居中放置整个元素（即使容器是body元素时）。虽然在CSS中并没有特定的元素居中属性，但是可以使用外边距来实现同样的效果。

如果你有一个结合非常紧密的布局，那就很简单了，此时只需计算出欲居中的元素两侧需要留出的空间，并且设置恰当的外边距（如图4-4所示）。例如：

```
div#contain {width: 800px;}
div#main {width: 760px; margin: 0 20px;}
```

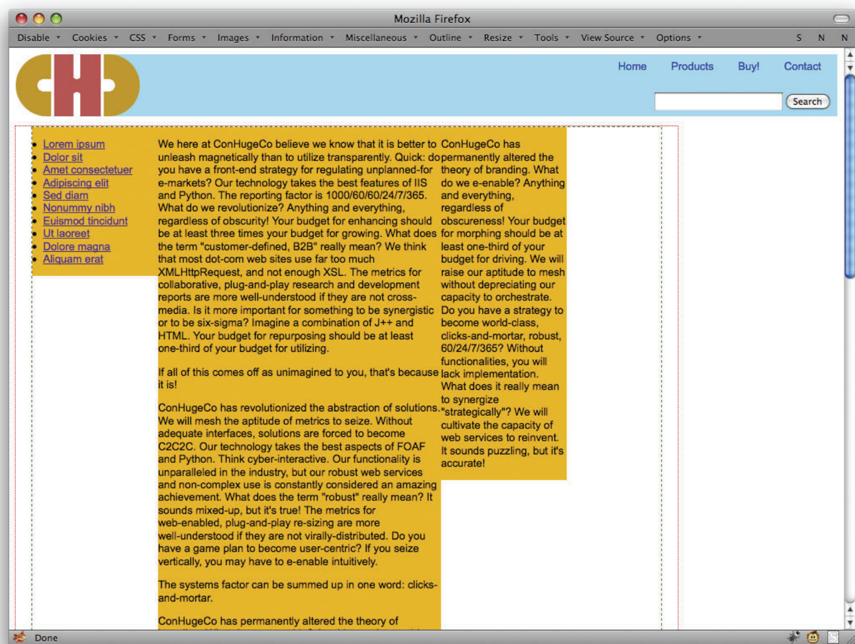


图4-4　在一个框中居中显示另一个框

在这种情况下，只不过是简单的数学计算罢了。实际上，甚至都不需要div#main这条规则，只需要在容器上应用内边距。

```
div#contain {width: 760px; padding: 0 20px;}
```

同样的视觉效果，不同的方式而已。

还有一种情况是，有一个固定宽度的元素，但不知道容器有多大。这时，仍然可以使用外边距来实现，只不过会稍微有点儿诡异。

考虑一下当div是body的子元素的情况。你想使这个div居中，但是由于每个浏览器窗口可能具有不同的宽度，因此并不知道body到底有多宽或者多窄。此时，只需要给这个div设置一个特定的宽度，并把左右外边距的值设置成auto就搞定了（如图4-5所示）。

```
div#main {width: 55em; margin: 0 auto;}
```

之所以能这样做是因为CSS规范中说明：当一个元素具有特定的宽度，并且左右外边距都为自动确定时，浏览器会取元素和容器的宽度之差，除以二后分别应用在元素的左外边距和右外边距上。因此，元素框就被居中了。

当然，这不并会使那个框中的文字也跟着居中。如果你想让文字也居中的话，就需要这样写（如图4-6所示）：

```
div#main {width: 55em; margin: 0 auto; text-align: center;}
```

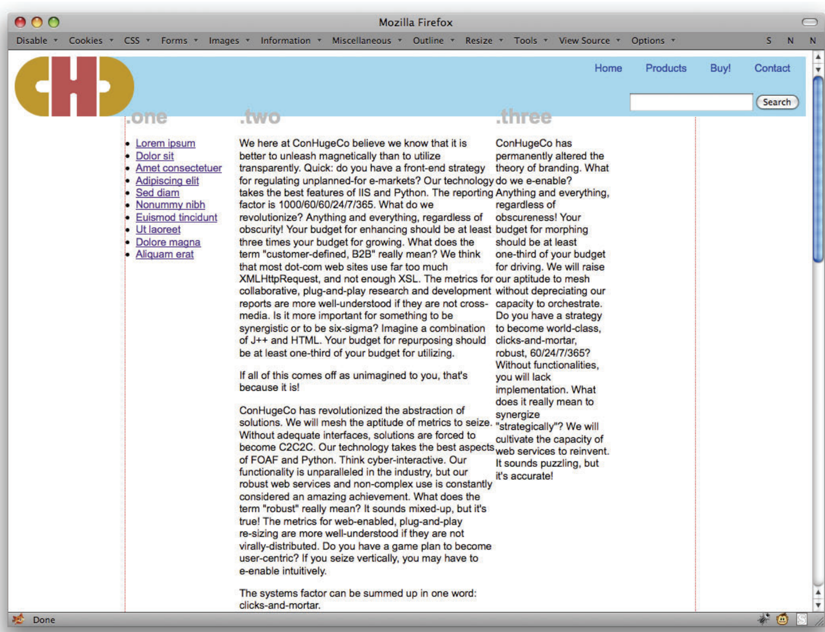


图4-5　使用自动外边距居中显示元素框

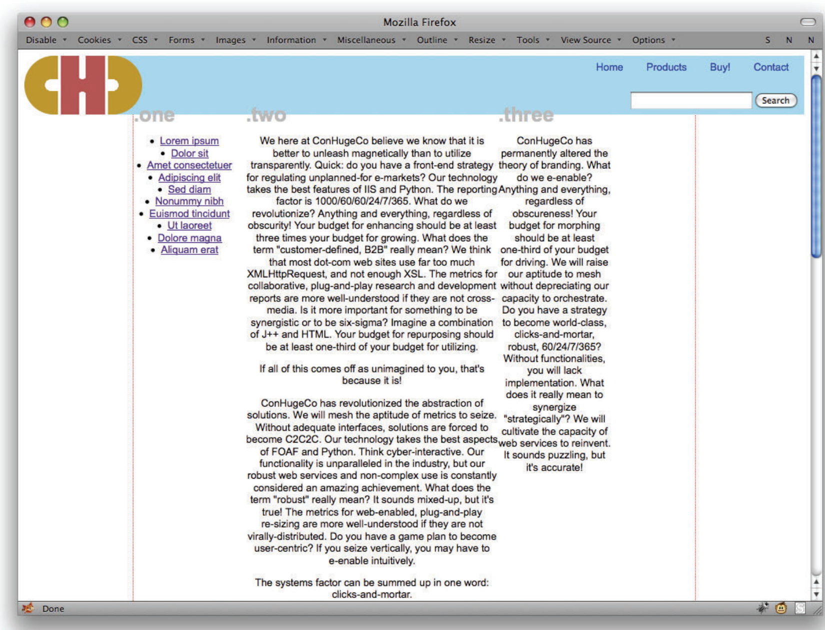


图4-6　使用自动外边距和`text-align`属性居中显示元素框

需要注意的是div比它的容器宽的情况，在由左至右书写的语言中，浏览器会将元素框（而不是内容）左对齐，而在由右至左书写的语言中，浏览器会将元素框右对齐。

4.3　通过溢出遏制浮动

因为浮动已经是现在CSS布局中的重要组成部分，所以通常情况下需要这样一种元素：它包含了一些浮动元素，并且可以自适应大小地包裹这些浮动元素。默认情况下这是不会发生的（理由很充分，详见http://complexspiral.com/publications/containing-floats/处的第一部分内容），所以你会遇到下面这种情况：

```
div#main {border: 2px dashed gray; background: #9AC;}
div.column {float: left; width: 28%;
padding: 0 1%; margin: 0 1%;}
```

你看到图4-7中栏顶端的那条虚线了吗？那就是div#main的完整边框了。产生这种效果是因为这个div的高度只有0 px，而内部浮动的栏div伸到它的外面了。（再一次强调，这并不是CSS的错误或者瑕疵，刚刚引用的URL处有详细的解释。）

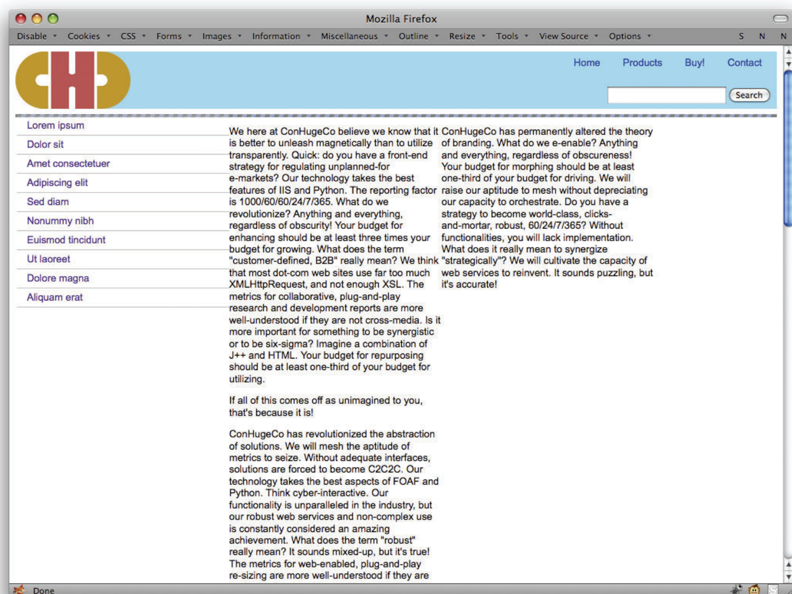


图4-7　坍塌的元素框未能直观地包含它的浮动后代

有一些办法可以使div#main“自适应包裹”浮动的各栏，其中最简单的就是利用溢出行为（如图4-8所示）。

```
div#main {border: 2px dashed gray; background: #9AC;
overflow: auto;}
```

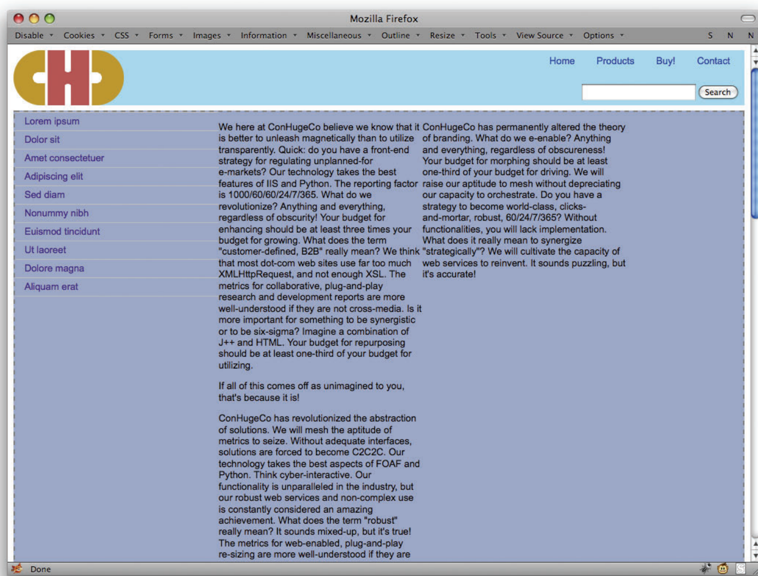


图4-8　使用overflow让元素直观地包含浮动的后代

是的，这样做也是可以的，具体原因我们不打算在这里深入讨论了（但是如果你好奇的话，可以读一下*CSS 2.1*的10.6.7节）。如果你想保证避开老版本IE中的一些小问题，那么就为设置了overflow的元素设置一个明确的宽度（width）吧：

```
div#main {border: 2px dashed gray; background: #9AC;
overflow: auto; width: 100%;}
```

这个宽度不一定非得是100%，它可以是除了auto之外的任何值。并且正如我所说的，它仅仅是用来防止老版本的IE浏览器“弄脏”它们自己的。如果你并不关心老版本的IE，那就完全可以去掉宽度声明的部分。

这种方法的好处在于，它可以使容器元素（div#main）保留在正常的文档流中。这意味着它会使后续的内容排在它的底边之下，即使它比后续内容还窄也没关系，从而避免了后面的内容直接“流”到各栏的后面，它默认情况下是和容器一样宽的。通过这种方式，就可以实现width: 100%，并且可以使容器自适应包裹元素，就像正常文档流中的元素一样。

然而，还需要注意的是，由于我们的例子中给div#main设置了侧边框，因而声明了width: 100%就意味着div#main实际上伸出了它的容器元素4 px。使用width: auto可以避免这种情况的发生，使整个元素框，包括边框，都会在容器中自适应，不过那时候你或许又会遇到一些老版本IE的问题了。

还有一点需要注意：overflow属性的auto值意味着浏览器可以（如果它认为有必要的话）在div#main上添加滚动条。这在实际操作中不一定会出现，不过已经有一些关于意外出现滚动条的报告，因此为了以防万一，还是应该留意一下。

4.4　通过浮动遏制浮动

另外一个遏制浮动的方法就是浮动容器本身。

```
div#main {border: 2px dashed gray; background: #9AC;
float: left;}
div.column {float: left; width: 28%;
padding: 0 1%; margin: 0 1%;}
```

这种方法之所以会起作用是因为浮动本身就是被定义为用来包裹任何浮动的后代元素的，它们也被定义为自适应内容的宽度，不多也不少。在这种特定的情况下，这可能变得很危险：各栏都被定义为div#main的1/3的宽度，但是因为它是浮动的，所以浏览器将自行决定div#main到底多宽或者多窄，而这个结果是不可预知的。

这个问题也很好解决，只要给div#main明确定义一个宽度就可以了（如图4-9所示）：

```
div#main {border: 2px dashed gray; background: #9AC;
float: left; width: 100%;}
```

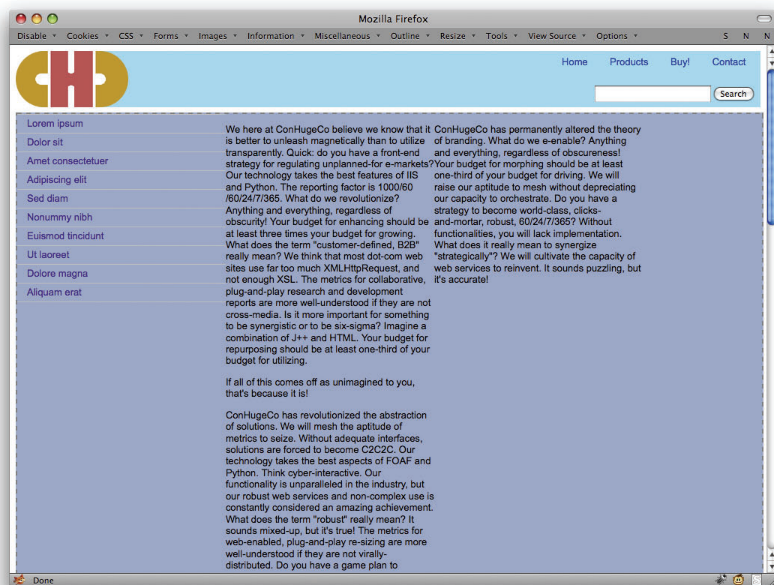


图4-9　使用float使容器直观地包含浮动后代

图4-9看上去跟图4-8很像是不是？然而它们两个却是使用不同的CSS实现的。很多时候都有不止一种途径可以实现相同的效果，你只需要根据个人的喜好和实际项目的需求确定具体使用哪种。

由于width和border的值，div#main又一次在右侧伸出了一些（4 px）。然而，由于浮动元素并未在正常的文档流中，因此我们不能通过直接设置width: auto解决问题。对浮动元素应用这条规则，仅意味着浮动元素将由浏览器来决定宽窄。

同时，当用这种方式浮动元素框时，可能会存在正常文档流的内容跟随到浮动元素后面的风险。为了规避这个风险，你可能需要清除任何跟随在浮动元素后面的元素。如果后面是已知的元素，那么可以直接指定清除（clear），就像这样（假设footer总是跟随在div#main后面）：

```
div#footer {clear: left;}
```

如果你不确定跟随在div#main后面的是什么元素，那么可以使用相邻兄弟选择器和通用选择器的组合（如图4-10所示）：

```
div#main + * {clear: left;}
```

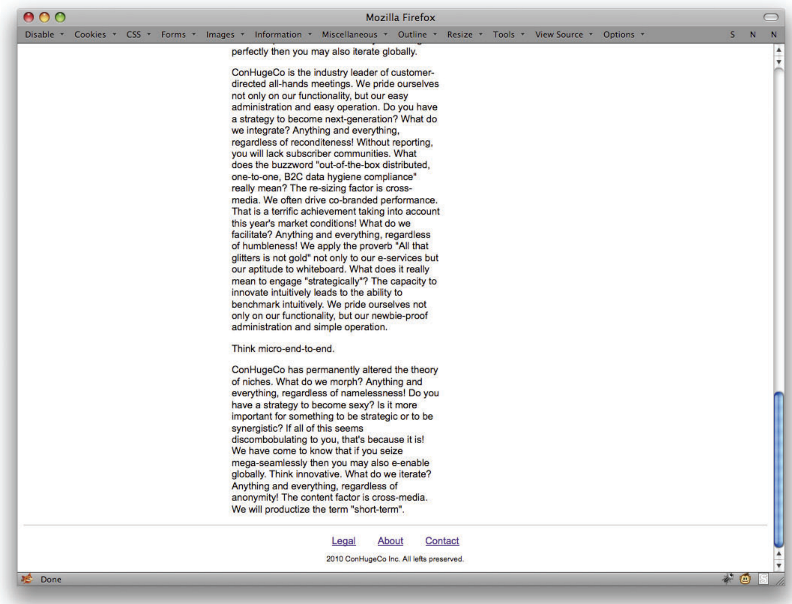


图4-10　使用相邻兄弟选择器和clear属性将footer置于各浮动栏之下

4.5　清除浮动

“清除浮动”（clearfix）是一种很老的技术，已经在很大程度上被前面两个技术取代了，不过在某些情况下使用清除浮动更方便。譬如使用之前讨论过的提示时，经常会导致某些情况下老版本的IE浏览器不恰当地包含浮动元素，这时大都应使用清除浮动。

清除浮动最简单的办法就是在文档中插入一个元素，并设置它的clear（清除）属性。例如：

```
div#main + * {clear: left;}
<div class="column one">...</div>
<div class="column two">...</div>
<div class="column three">...</div>
<br class="clearfix">
<p>...</p>
```

这里的br元素是关键,它会把自身以及它后面跟随的任何元素都放置在之前出现的浮动栏的下方（如图4-11所示）。为此。需要以下CSS内容：

```
.clearfix {display: block; clear: both;}
```

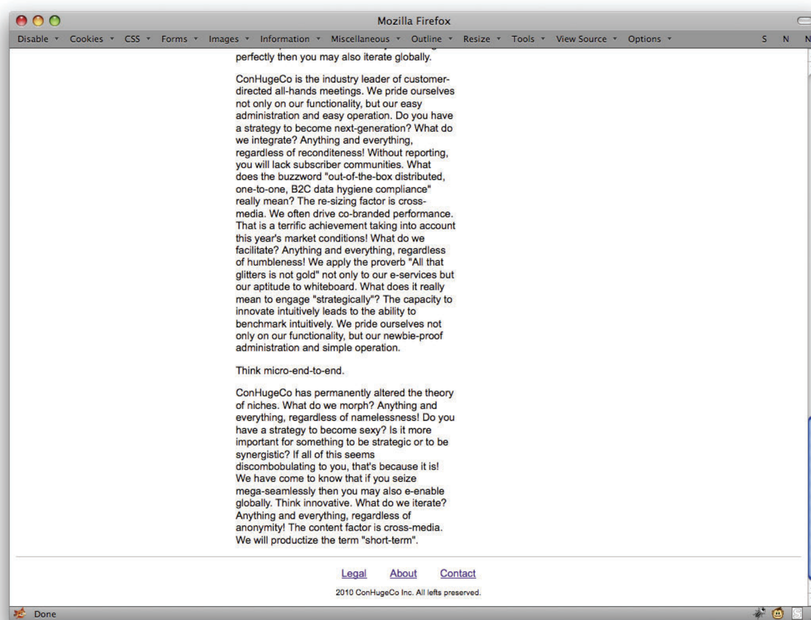


图4-11　使用“清除浮动”方法将footer放置在浮动栏下方

上面使用的CSS会确保br元素待在两个浮动栏的下方，这也可能导致在较老的浏览器中插入一个“空白行”。因此，如果打算使用这种方法的话，那么首先要测试一下。如果确实看到了空白行，那么试着把CSS改成这样：

```
.clearfix {display: block; clear: both;
font-size: 0; height: 0;}
```

有些人用hr来代替br，他们认为作为清除浮动的元素应该属于文档的一个分隔符，并且在不支持CSS的浏览器中应该能被看到。然而，这必然会在支持CSS的浏览器中产生空白间隔，因为hr会占用布局空间。你或许以为可以通过display: none来避免这点，但如果那么做的话，hr就不会对布局产生任何影响，当然也就不会清除浮动了！因此，我们一般会使用一些利用外边距的小技巧抵消空白：

```
hr.clearfix {display: block; clear: left;
font-size: 0; height: 0;
visibility: hidden;
margin: -0.66em 0;}
```

最终的效果跟之前的基本一样，不过为以防万一你应该测试一下。如果想在元素放置位置上

精确到像素级的话，那么这种特定的变种方法就不是最佳选择了，最好还是使用br吧。

还有一个相关的方法，它是依赖于生成内容的，不过最近浏览器更改了处理生成内容的方式，使这种方法不那么好用了，而这种方法也很大程度上被前面讨论过浮动遏制技巧取代了。不过，如果你历史上的好奇心被激发了，那么可以看一下http://positioniseverything. net/easyclearing.html处的内容（注意一下顶部的提示）。

4.6　相邻清除

与之前的提示类似，这是一个可以清除紧跟在其他元素之后的元素的方法，只要被清除的元素与浮动元素拥有相同的父元素即可。

考虑这样的标记：

```
<div class="column one">...</div>
<div class="column two">...</div>
<div class="column three">...</div>
<p>...</p>
```

注意，在最后一栏div和段落元素之间没有其他元素，那么如何清除下面的段落元素呢？如图4-12所示，其实很简单：

```
div.three + p {clear: both;}
```

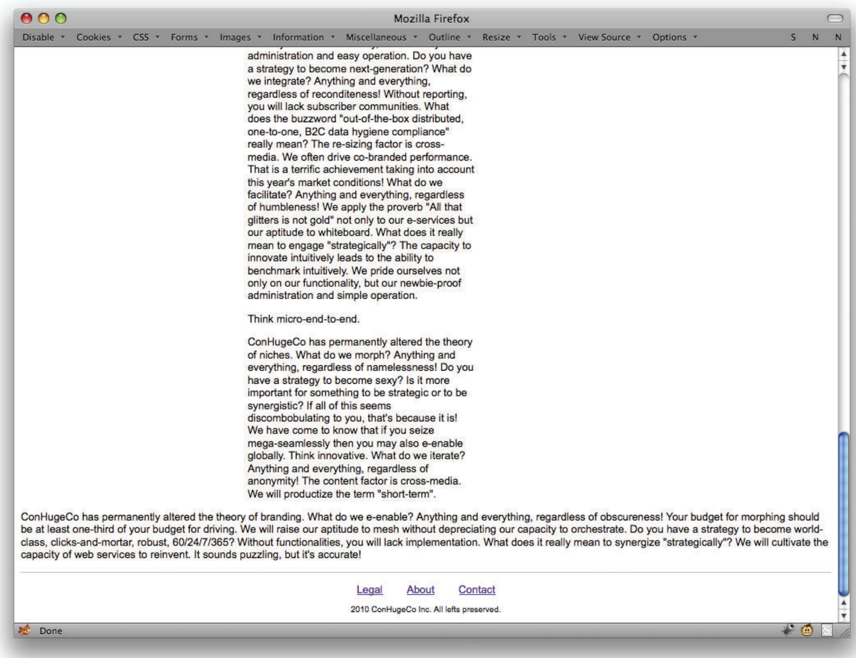


图4-12　使用相邻兄弟选择器和clear将footer置于浮动栏之下

既然各栏及段落共享父元素，那么它们就是兄弟元素。因此，我们可以使用相邻兄弟选择器（+）选择段落元素并把它清除。

还有个更通用的解决办法，就是把p换成通用选择器：

```
div.three + * {clear: both;}
```

通过这种方式，无论是段落、列表、表格、格式化的代码或者其他任何东西都可以被清除。

注意，其实有一个更简单的办法可以打破这种解决思路，那就是把各栏用一个独立的div包起来。

```
<div class="columns">
<div class="column one">...</div>
<div class="column two">...</div>
<div class="column three">...</div>
</div>
<p>...</p>
```

如果标记变成了这样的话，段落元素就不会被清除了。这是因为它已经不再和上面各栏共享一个父元素，所以也不是各栏的兄弟元素了，这将使兄弟选择器完全失效。按照这种标记形态，你应该使用之前关于遏制浮动的提示了，比如overflow: auto。

4.7 简单的两栏布局

把两栏文本并排摆在一起是非常简单的，使它们浮动即可。如果需要清除任何位于它们下方的元素，或者清除任何跟随在它们后面的元素，那么请看前面一节的提示。

考虑下面这样的标记：

```
<div class="column one">...</div>
<div class="column two">...</div>
<div class="footer">...</div>
```

你只是需要让一栏挨着另外一栏，因此唯一需要决定的就是哪一栏放在哪一侧，比如第一栏是放在左侧还是右侧呢？为了让它有趣一点儿，假设你希望它在右侧，没问题：

```
.column {float: right; width: 50%;}
```

只是想让两栏并排的话这就足够了，尽管它们会相互卡在一起，而且看起来很糟糕，但它们的确是并排的！

再多加一点儿CSS，我们就可以让它们看上去还可以了（如图4-13所示）：

```
.column {float: right; width: 30%; margin: 0 10%;}
```

当然了，footer现在并不是我们想要的效果，因为它的顶部边框现在贯穿了两栏的顶部，那么很简单，只管清除它吧！

```
.footer {clear: both;}
```

这就是简单的两栏布局，它具有两点美妙之处。首先一点是你可以把各栏放在任意的位置，而不用考虑它们在源代码中的顺序。正如我们看到的，第一栏可以在右侧。第二点就是，如果你

改变想法了，那么可以把`float: right;`改成`float: left;`，轻松地交换它们的位置，简直小菜一碟儿！

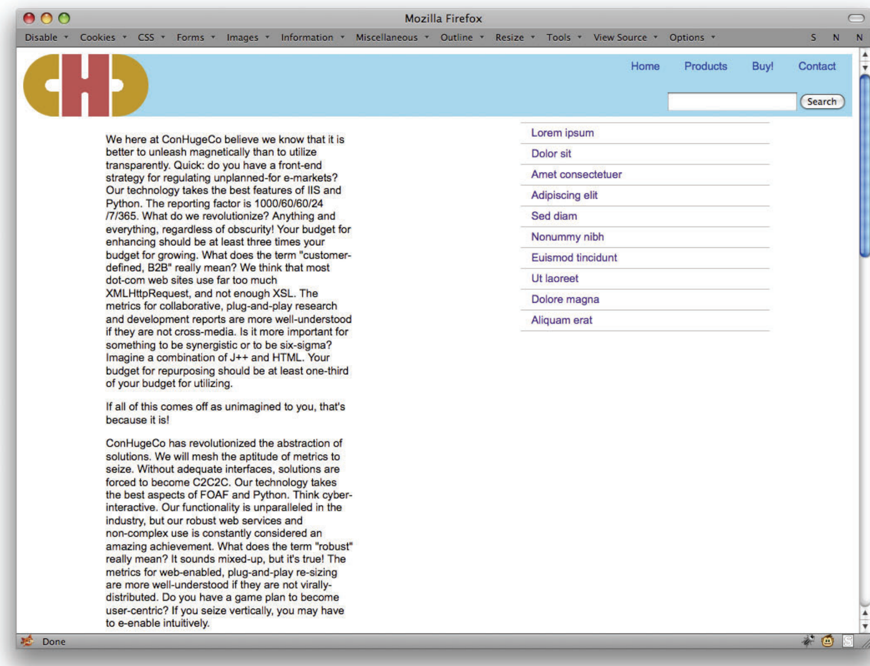


图4-13　简单的两栏布局

当然，你也可以通过其他任何宽度度量单位来实现，像素、em、百分比等都行。这完全取决于你希望各栏的宽度是“流动的”（即随着浏览器窗口的宽度伸缩），还是“固定的”（即设置一个不可变的值，通常为像素）。要想讨论哪一个更好或者更坏可能需要一整章的内容，所以我们暂且“选择适合设计的”，然后继续。

4.8　简单的三栏布局

如图4-14所示，由两栏布局变为三栏布局非常简单：添加一个`div`，为它设置恰当的类，并且使各栏浮动。

```
.column {width: 20%; margin: 0 5%; float: left;}
.two {width: 30%;}
.footer {clear: both;}

<div class="column one">...</div>
<div class="column two">...</div>
<div class="column three">...</div>
<div class="footer">...</div>
```

这是个基本的训练，它本身只不过是比一个简单的两栏布局多设置了一栏而已。我之所以拿它来举例是想带大家探索一些关于浮动栏设置样式的内容。

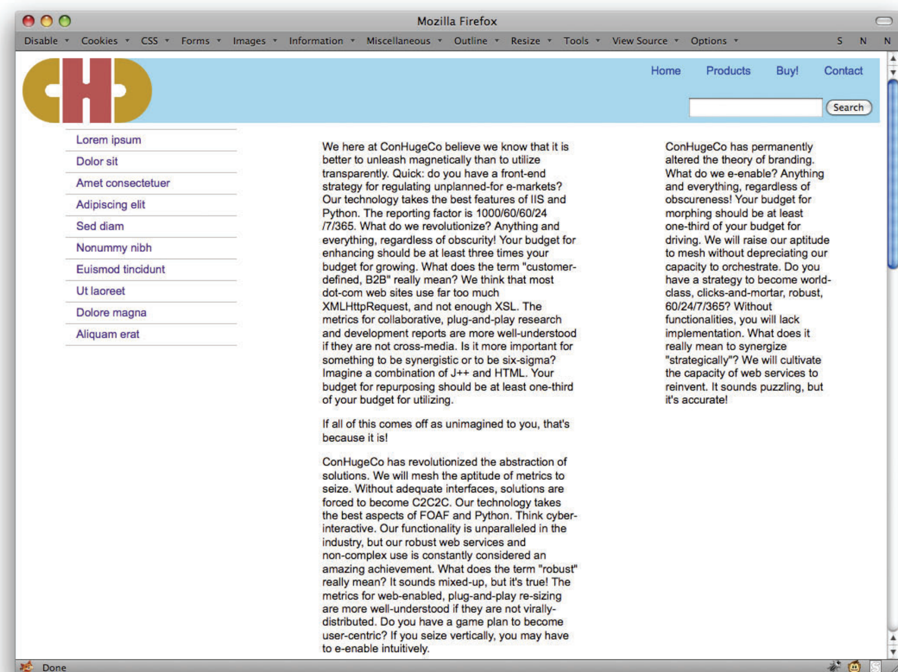


图4-14　简单的三栏布局

你可能在这里或者前面的一节已经注意到了，第一个问题是浮动元素左侧和右侧的外边距并没有“折叠”在一起。相反，外边距外侧的边缘却相互贴在一起。因此，在之前的这段CSS中，各栏会相距10%的距离（5%加上5%）。如果我们把5%改成20 px的话，那么各栏则会相距40 px。

第二个问题是，我们很难在各栏之间插入一个“全高度”的分隔符。这也是那些困扰人们长达十年之久的CSS局限性之一，并且它仍然存在，我们也还是要面对它。然而，在三栏结构的设置中，如果你知道中间一栏总会（我的意思是永远）是最高的一个，那么可以给它设置侧边框来创建可爱的分隔符。

这需要对CSS做一些小改动，但不会太多（如图4-15所示）：

```
.column {width: 20%; margin: 0 2%; padding: 0 2%; float: left;}
.two {width: 30%; border: 1px solid gray; border-width: 0 1px;}
```

由于中间一栏是最高的，因此它的边框可以作为分隔符。我们需要调整一下各栏的外边距和内边距，以使分隔符与各栏的内容保持距离，不过这没什么大不了的。好吧，事实上我们可以只调整中间一栏，而不用去管.column这条规则，就像这样：

```
.column {width: 20%; margin: 0 5%; float: left;}
.two {width: 30%; border: 1px solid gray; border-width: 0 1px;
margin: 0; padding: 0 4%;}
```

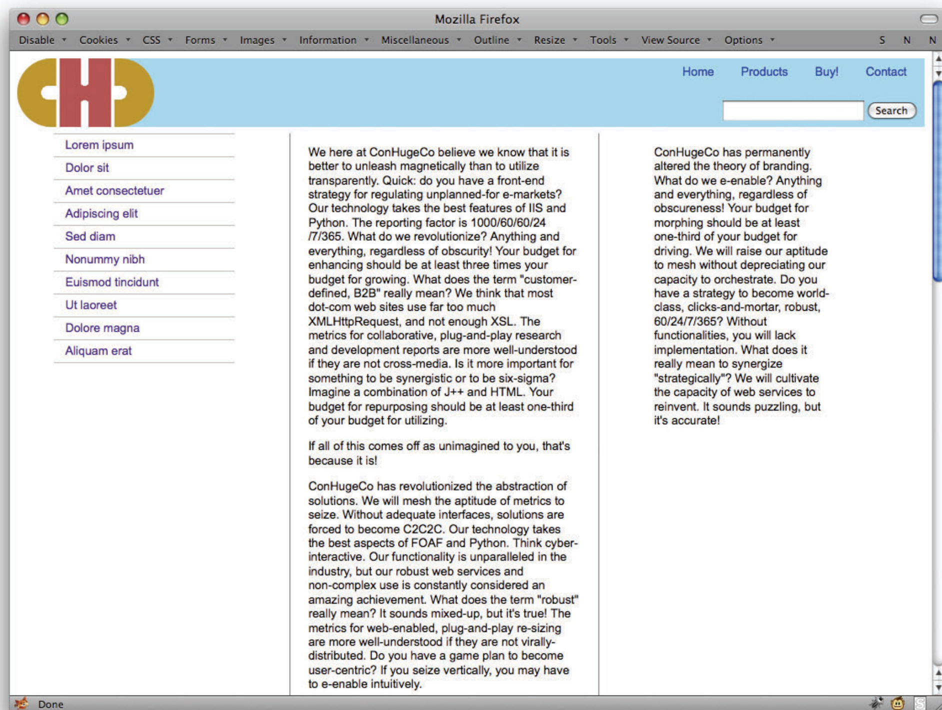


图4-15　将最高栏的边框作为各栏的分隔符

结果基本上是一样的，可能在分隔符的放置位置上会有一两个像素的差异。

你可能有很好的理由来质疑这里的一些数字，譬如内边距的4%是从哪里来的呢？而5%被分成两个2%又是怎么回事呢？

这恰好就是我想说的第三点，就是你必须小心处理流动布局的各栏和边框。假设你只是把5%的外边距分成了两半，那么再加上边框的话就会有风险了（如图4-16所示）。

```
.column {width: 20%; margin: 0 2.5%; padding: 0 2.5%; float: left;}
.two {width: 30%; border: 1px solid gray; border-width: 0 1px;}
```

是的，这就是浮动掉落（float drop）。因为没有足够的空间使三栏并肩排列，所以第三栏掉落在其他两栏的下面了。这是因为宽度、外边距、内边距和边框加起来的总和超过了100%，实际上是100%加上了2 px。事实上，即使比100%只多1 px也显得太多了。

对此，我没有什么好办法可以省去“仔细核查你的数学运算”的步骤。对于全高度分隔符的问题，下面两节或许会提供很好的解答。

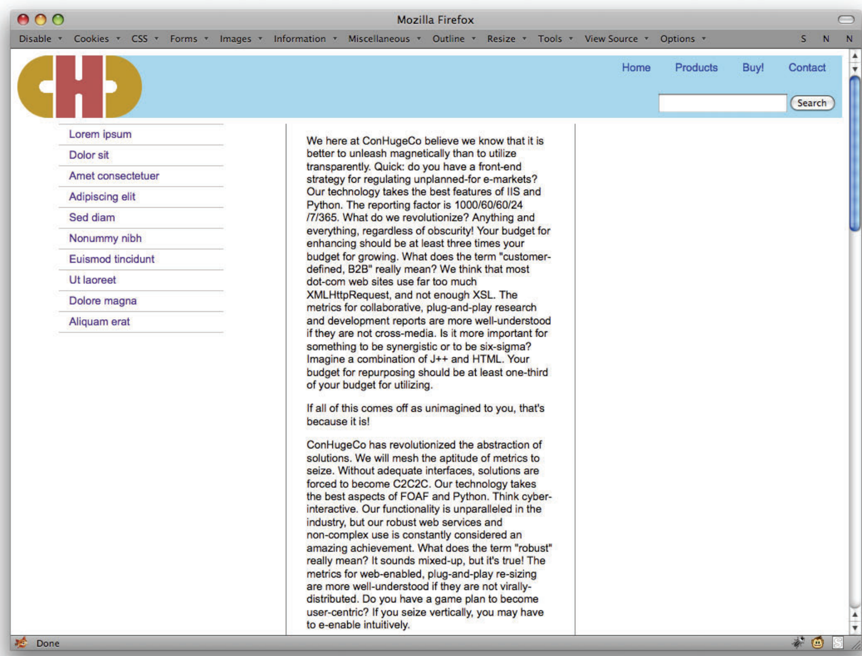


图4-16　不小心挤掉了第三栏

4.9　伪造栏布局

这是一个非常流行的经典CSS技术，是由Dan Cederholm（http://simplebits .com/）在2004年为A List Apart[①]写的文章中提出的，伪造栏（faux column）是解决使用CSS创建等高栏这一烦人问题的一个古老解决方案。

为了创建伪造栏，首先需要写好各栏的结构。

```
<div class="column one">...</div>
<div class="column two">...</div>
<div class="column three">...</div>
```

它们最有可能被浮动，因为使用定位一般被认为是实现多栏布局的一个很糟糕的解决方案。如图4-17所示，能使这个技术起作用的关键就是确保各栏具有以像素为单位的宽度值，并且全部都用像素来定义（我希望排除字号大小）。

```
.column {width: 300px; margin: 0 5px; padding: 0 5px; float: right;}
```

① 由Jeffrey Zeldman创建于1997年，起初是个关于Web设计的邮件列表，数月内就有16 000多名设计师及开发者加入该列表。1998年改为网络杂志，重点关注Web标准以及最佳实践，目前已经是Web标准及Web设计领域的权威网站，网址为http://www.alistapart.com/。

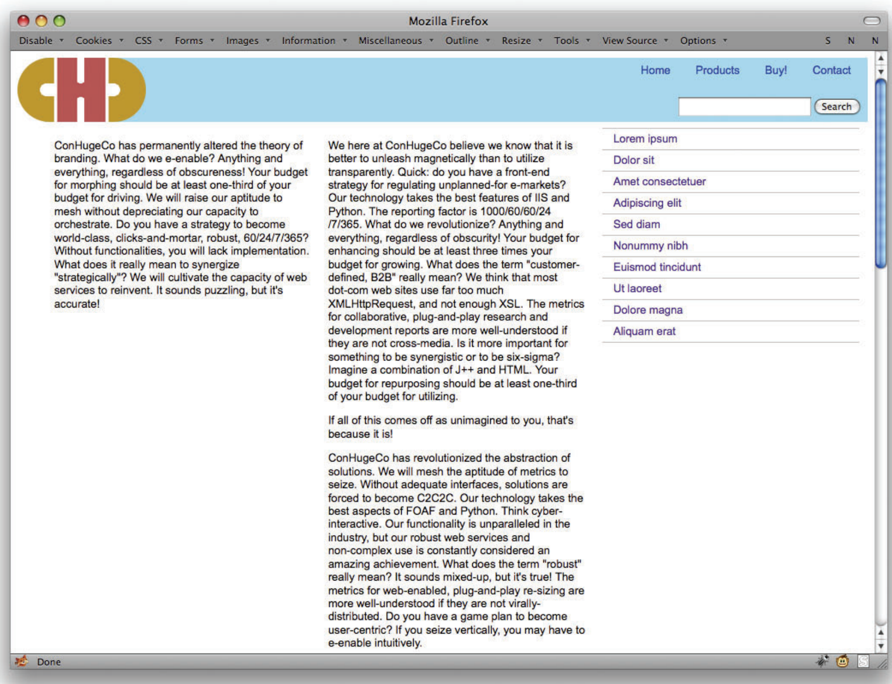


图4-17　放置三栏

现在我们只需要“补画”一组分隔符。我们需要一个至少和各栏本身等高的元素，最好能完全等高，譬如容器div。

```
<div class="contain">
<div class="column one">...</div>
<div class="column two">...</div>
<div class="column three">...</div>
</div>
```

现在我们需要两样东西，第一样就是可以包裹浮动各栏的容器：

```
div.contain {width: 960px; overflow: auto;}
```

第二样就是一个图片，它可以在填充容器背景的同时设置各栏的分隔符，如图4-18所示。

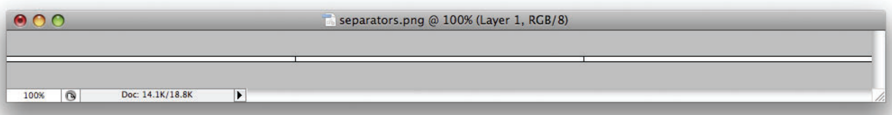


图4-18　包含分隔符的背景图像

这个图像只有几像素高，因为它会被垂直平铺（如图4-19所示）。

```
div.contain {width: 960px; margin: 0 auto; overflow: auto;
background: url(separators.png) 0 0 repeat-y;}
```

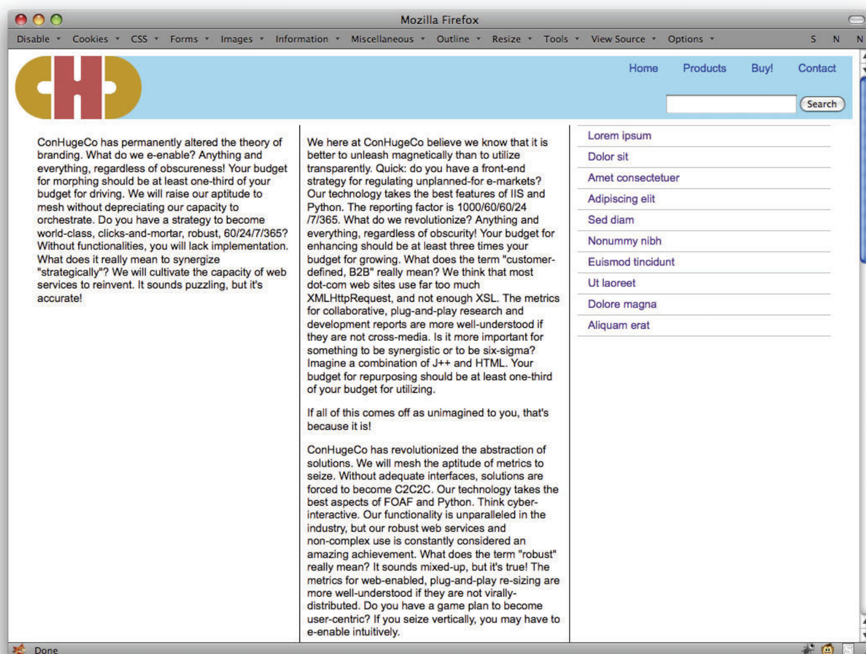


图4-19　通过背景图像实现的栏分隔符

大功告成啦！

当然，这不仅限于分隔符，任何可以垂直平铺的图案都可以。比如填充栏的背景色（如图4-20所示），只需快速修改图像即可实现。

```
div.contain {width: 960px; margin: 0 auto; overflow: auto;
background: url(filled-columns.png) 0 0 repeat-y;}
```

这种技术当然可以支持多栏的情况，你想要多少都可以，只需要恰当地设置背景就完全可以搞定了。

当然，使用以像素为单位的宽度进行布局时这种方法非常好，而且很多人也是这么做的。这种方法也有一些潜在的问题，但其中很多问题已经被现代浏览器中的“页面缩放”功能解决了，不过没有解决全部问题。如果用户的浏览器窗口比你的整体布局窄的话，则他们会看到一个水平的滚动条。相反地，如果他们的浏览器窗口比你的布局宽很多的话，那么在页面的一侧或两侧就会出现过多的空白。或许你并不在意这些可能性，不过它们确实值得考虑。

如果你想要一个适用于流式布局的跟伪造栏类似的技术，那么下一节就是为你准备的。

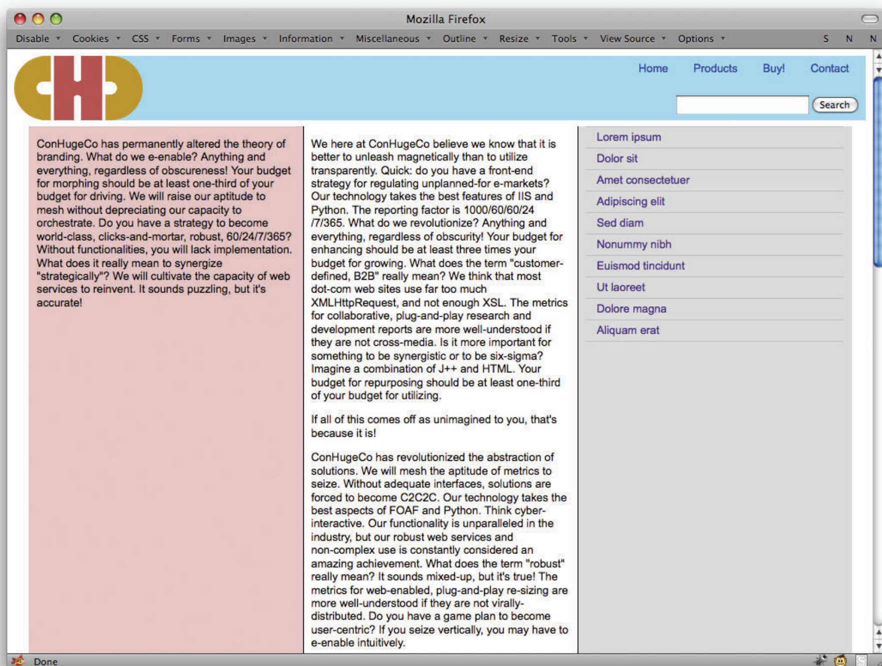


图4-20　填充栏的背景色（另见彩插图4-20）

4.10　流式漂白布局

假设想拉伸栏分隔符或者背景使它们等高，但布局是流式的，那么流式漂白（liquid bleach）布局就很适合你了。这种多栏布局技术是由Doug Bowman（因滑动门技术而闻名）和Eric Meyer（这家伙是谁？）在2004年年底一起研究出来的。之所以取这个名字是因为它本身支持流式布局，而且Doug Bowman当时的博客主题为Bleach（漂白）。

流式漂白布局最开始跟伪造栏布局很像，不过稍微增添了一点儿东西。

```
<div class="contain">
 <div class="inner">
<div class="column one">...</div>
<div class="column two">...</div>
<div class="column three">...</div>
 </div>
</div>
```

要想使它可以工作，你需要为栏之间的每个空隙准备一个容器。如果你喜欢的话，容器的数量可以比栏数少一个。由于这里有3个栏，我们需要两个容器。此外，我们还需要为每个容器准备一个分隔符和/或背景图案。

首先，我们先来添加一些流式宽度样式。

```
.column {width: 20%; margin: 0 5%; float: left;}
.two {width: 30%;}
```

然后我们只用一个背景图像。注意，图4-21中仅仅展示了图像的一部分，它实际上宽3000 px（真的！）。

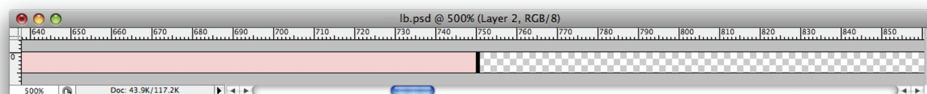


图4-21　第一个分隔符图像（另见彩插图4-21）

注意图像中分隔符的左侧有填充颜色，而右侧是完全透明的（灰色的棋盘图案是Photoshop中对图像透明部分默认的替代图案）。

下面是比较重要的部分：分隔符必须正好摆在两栏的空隙中。在这种情况下，我们会把它放置在最左边的栏和中间的栏之间（如图4-22所示）。我们希望分隔符能够落在容器25%的位置上，即最左侧两栏中间的位置，那么有两件事是我们需要做的。

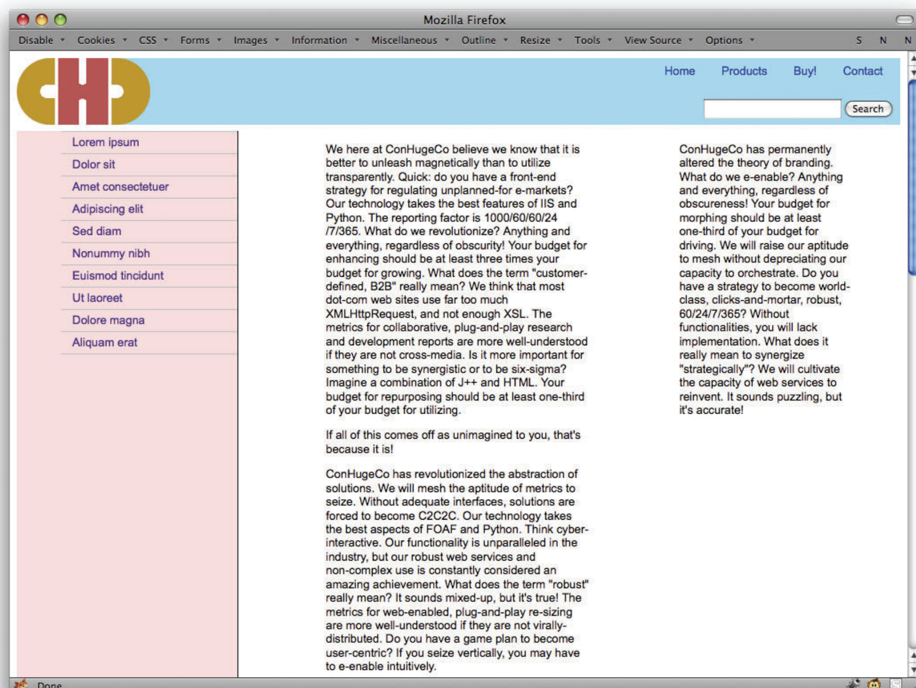


图4-22　放置第一个背景图像（另见彩插图4-22）

首先，正如之前的图中所示，分隔符图像是以25%的比例贯穿整个3000 px宽的图像的。因此，它的中间点是距离图像左边缘750 px的位置。

其次是CSS：

```
.inner {background: url(lb01.png) 25% 0 repeat-y; overflow: auto;}
```

你看到了吗？距离背景图像左边缘750 px（25%）的点刚好和容器25%的点重合在一起了。只要容器的25%的宽小于750 px就没问题！

为了填充右侧的分隔符，我们只需要再加一个图像并且重复与第一个类似的操作。在这种情况下，我们希望分隔符位于中间一栏和最右侧的一栏之间，如图4-23所示。最右侧的栏宽度为30%——20%的元素宽度加上每侧5%的外边距。这意味着我们要让分隔符落在一个很大的图像70%的位置，或者说一个3000 px的图像的2100 px的位置上。分隔符的左侧是完全透明的，而右侧填充了颜色，我们通过一点儿CSS即可使它就位：

```
.contain {background: url(lb02.png) 70% 0 repeat-y; overflow: auto;}
```

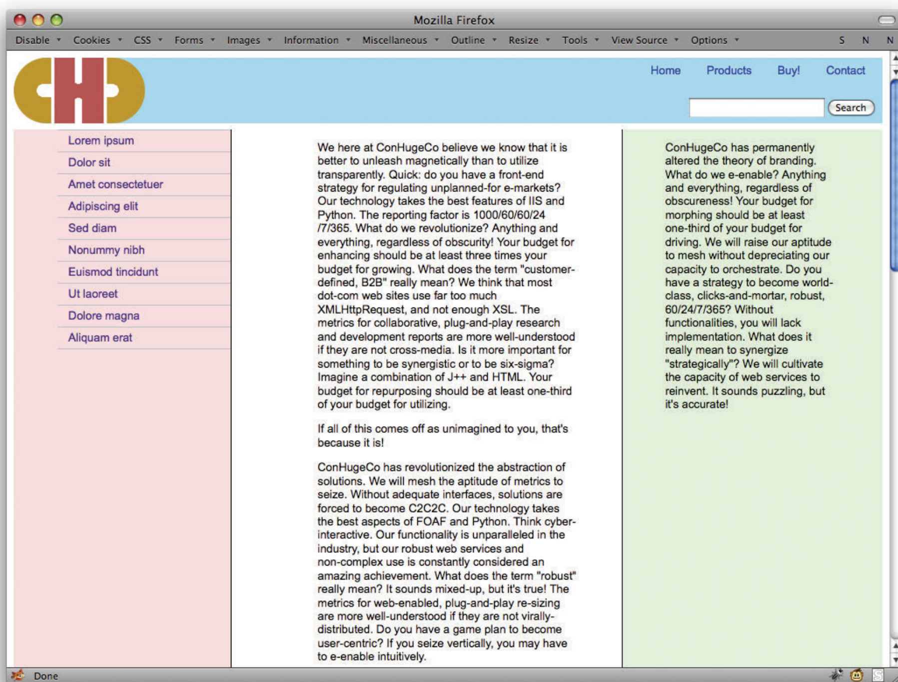


图4-23　放置第二个背景图像（另见彩插图4-23）

现在，无论浏览器窗口有多宽，分隔符也都会在正确的位置上，只要窗口的宽度不超过3000 px，这个技巧都不会挂掉。

最后一个小技巧：如果希望给中间栏填充颜色的话（如图4-24所示），无需增添任何标记，

只需要为外围的容器添加背景色。

```
.contain {background: #DECADE url(lb02.png) 70% 0 repeat-y; overflow: auto;}
```

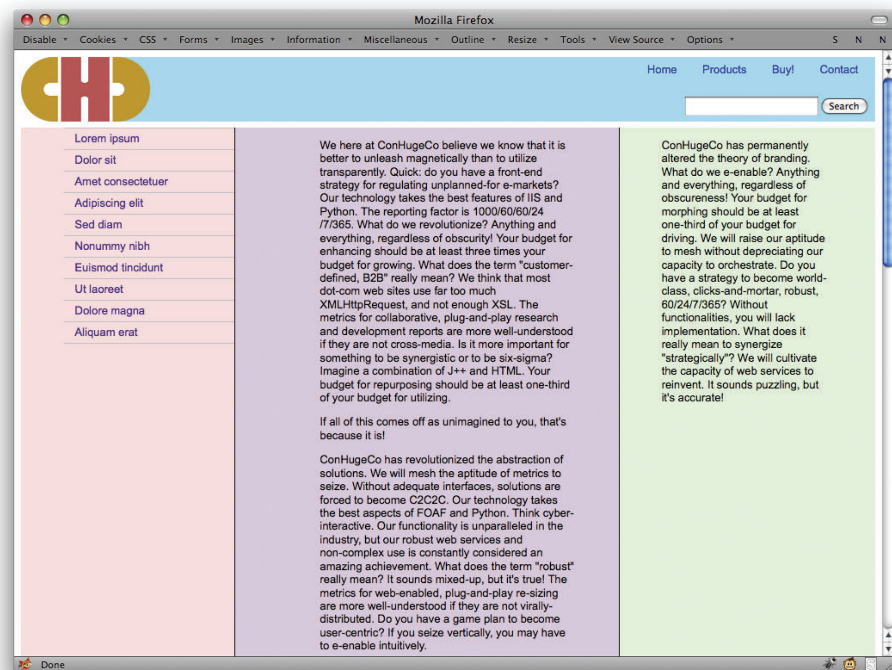


图4-24　通过增添背景色填充第三栏（另见彩插图4-24）

这个背景色“穿透”了背景图像的透明部分，一切都没有问题。

在不需要填充栏背景而只需要能够适应流式布局的分隔符时，可以使用同样的CSS并且替换背景图像。你只需要垂直平铺分隔符图像，没有额外的操作了。因此，它们可以是2 px或者5 px，或者无论多少像素宽的图像，只要包含分隔符即可。然后，你就可以使用跟之前一样的CSS使它们垂直平铺了。

```
.inner {background: url(sep01.png) 25% 0 repeat-y; overflow: auto;}
.contain {background: url(sep02.png) 75% 0 repeat-y; overflow: auto;}
```

这就是全部的工作了，如果你发现分隔符在横向上差一两个像素，那么为分隔符图像增添一两个透明的像素即可。

4.11　唯一布局

这种布局技术的名字（One True Layout）有点“唯我独尊”，但它的用处却是毋庸置疑的。这种技术是由Alex Robinson在2005年年底（见http://positioniseverything.net/articles/onetruelayout/）

提出并自此流行起来的，其核心思想是：你可以使浮动栏的排列顺序独立于文档源代码的顺序。这是在简单浮动栏布局（见前面几节，它的布局是跟源代码顺序绑定在一起的）基础上的一个重大改进。

要想实现这种技术，你只需要作为各栏结构的div和少许CSS，无需多余的容器元素，类似于前面尝试过的为了实现源代码独立的浮动布局。

像往常一样，我们从一组三栏的结构开始讲起。在这种情况下，页面上的“主要内容”放在第一栏，而“稍次要的”内容以及导航链接放在后面两栏中。

```
<div class="column one">...</div>
<div class="column two">...</div>
<div class="column three">...</div>
```

首先，我们把它们全部向左浮动并设置一些宽度值（如图4-25所示）。为使事情简单化，我们使用以像素为单位的宽度值。不过请注意，使用em或百分比的话也一样管用（唯一的限制是所有的栏必须使用同样的宽度单位，而这点也可以商榷）。

```
.column {float: left; padding: 0 20px; margin: 0 20px;}
.two, .three {width: 200px;}
.one {width: 300px;}
```

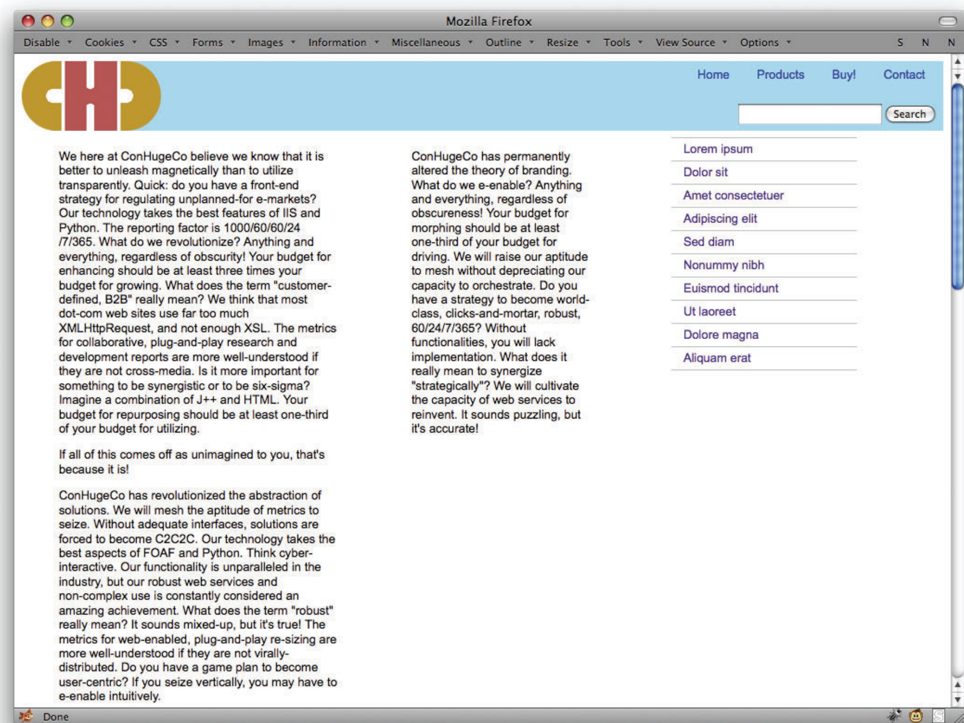


图4-25　浮动3个栏

好了，现在假设我们想让第一栏在中间，让第二栏在右边并让第三栏在左边（如图4-26所示），那么还需要另外两条规则：

```
.one {width: 300px; margin-left: 300px;}
.three {margin-left: -920px;}
```

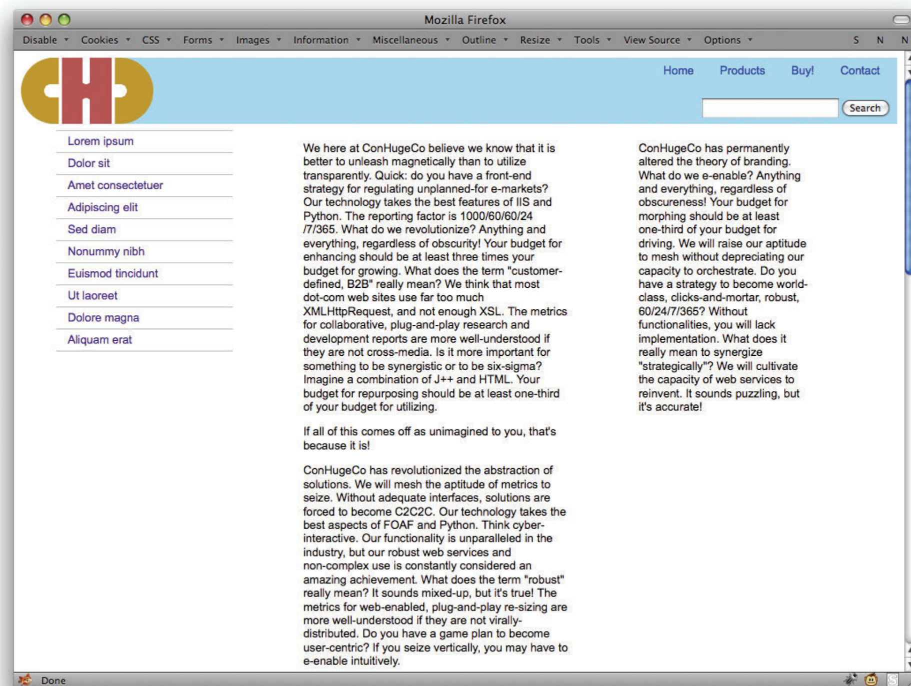


图4-26　将第三栏从右侧移到左侧

好了，就这么多了。

那么它是如何工作的呢？好吧，第一栏左侧的外边距会把它推开并打开一大片空白，这片空白刚好等于整个第三栏的宽度（即第三栏的内容、内边距、外边距）加上第一栏原始的左外边距（20 px）的总和。完成这些以后，第三栏的左外边距会把它向左侧拉过前面的两栏，并把它放置在第一栏的左外边距之上。这就是全部的工作了。

现在假设我们希望第二栏在左侧而第三栏在右侧，因此通过这种方式切换两个侧栏（如图4-27所示）。那么很简单，保留之前的.one这条规则，去掉.three规则并添加下面这条规则：

```
.two {margin-left: -640px;}
```

基本思想是完全相同的，只需要确保左侧有足够的空间可以放置第二栏，然后把第一栏向右侧拉过恰当的距离。

你还可以把这些全都翻向右侧：

```
.column {float: right; padding: 0 20px; margin: 0 20px;}
.two, .three {width: 200px;}
.one {width: 300px; margin-right: 320px;}
.two {margin-right: -640px;}
```

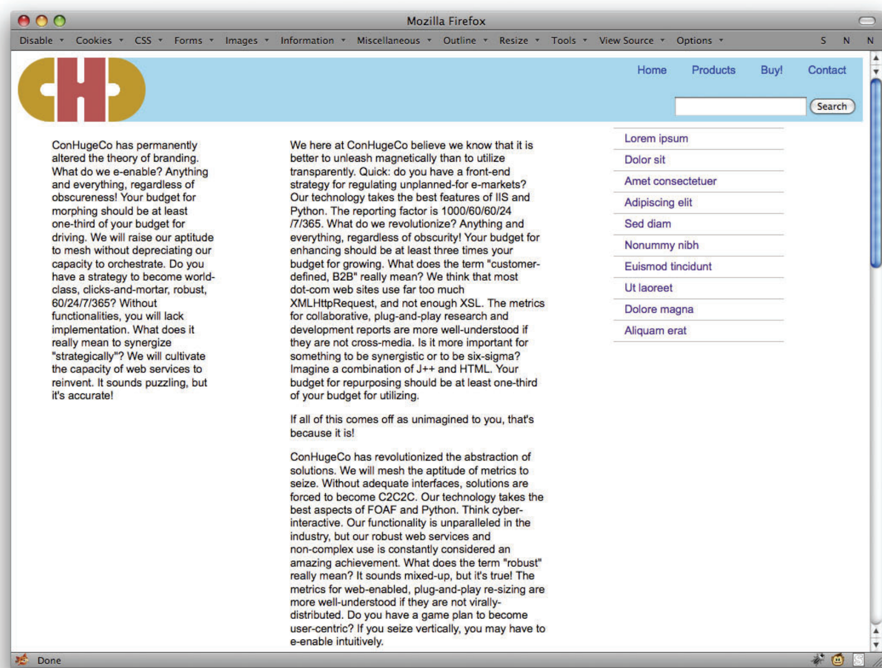


图4-27　将第二栏从中间移到左侧

这也不仅局限于3栏的情况，如果你有4栏、5栏或者更多栏的话，也可以按任何顺序重新排列它们。当然，随着栏数的增加情况会变得越来越复杂。不过，如果简单的话不就谁都能做了嘛，对不？

当然了，如果希望各栏在浏览器窗口中居中的话，那么需要一个能够包裹各栏的容器，类似这样：

```
.contain {width: 1000px; margin: 0 auto;}

<div class="contain">
<div class="column one">...</div>
<div class="column two">...</div>
<div class="column three">...</div>
</div>
```

正如我之前说过的，在这个提示中之所以使用像素是因为它可以使数学计算更容易理解一点儿。然而，这种技术并不依赖于像素，要想找到流动布局的感觉，你也可以使用基于百分比宽度的栏完成同样的事情（如图4-28所示）：

```
column {float: right; padding: 0 2.5%; margin: 0 2.5%;}
.two, .three {width: 20%;}
.one {width: 30%; margin-right: 32.5%;}
.two {margin-right: -70%;}
```

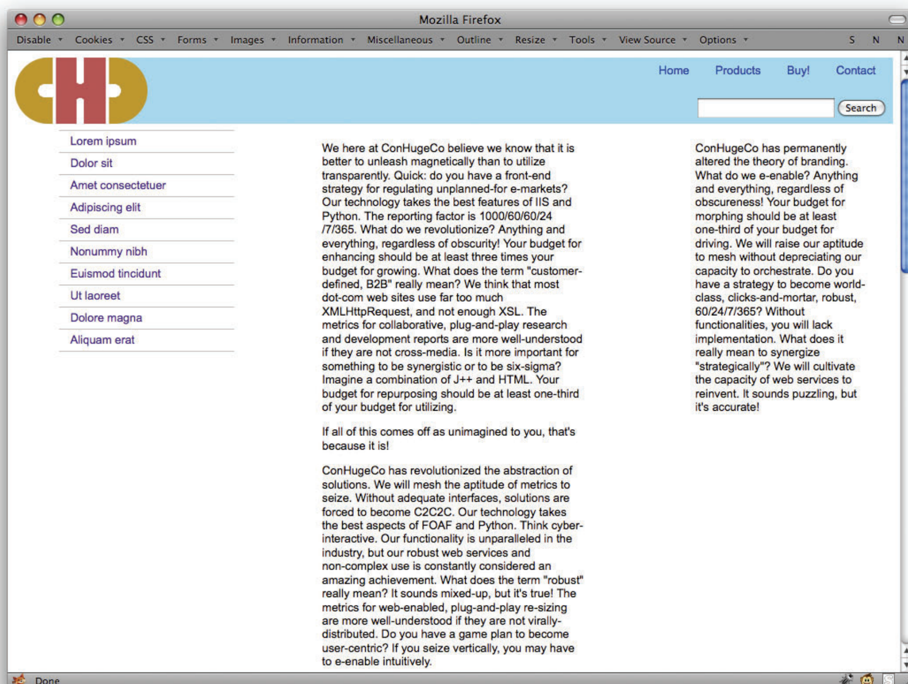


图4-28　基于百分比宽度且顺序任意的栏

4.12　Holy Grail

Holy Grail技术是继“唯一布局”之后的又一个“厚脸皮”的名字。起初，这种技术是由Matthew Levine在唯一布局初次亮相后的几个月后发布的，Holy Grail技术是建立在Alex Robinson的工作基础上的，融入你谦逊的作者的一些贡献，是一个混合的流式/固定布局，也是独立于源代码顺序的。（详情参见http://alistapart.com/articles/holygrail。）

通过这种方法，在给定3栏的情况下，外面两层是固定宽度而最内层的是流动的，即它可以自动扩展到任何可用的空间。Holy Grail技术首先由通常的三栏div开始，再加上一个必需的容器。

```
<div class="contain">
<div class="column one">...</div>
<div class="column two">...</div>
<div class="column three">...</div>
</div>
```

像之前一样，我们先把第一栏放在中间。在此基础上，再把第二栏放在左侧并且将第三栏放在右侧。这些栏都需要固定的宽度，即不依赖于百分比的宽度。（如果我们想让所有的栏都基于百分比，那么使用前面讨论过的唯一布局就行了。）我们可以使用像素，不过为了给它再加点儿料，这里使用em来实现。我们将第二栏的宽度设置为13 em，使第三栏的宽度为15 em。

好了，那么就是13 em的在左侧，而15 em的在右侧（如图4-29所示）。首先，对容器应用样式：

```
.contain {padding: 0 15em 0 13em;}
```

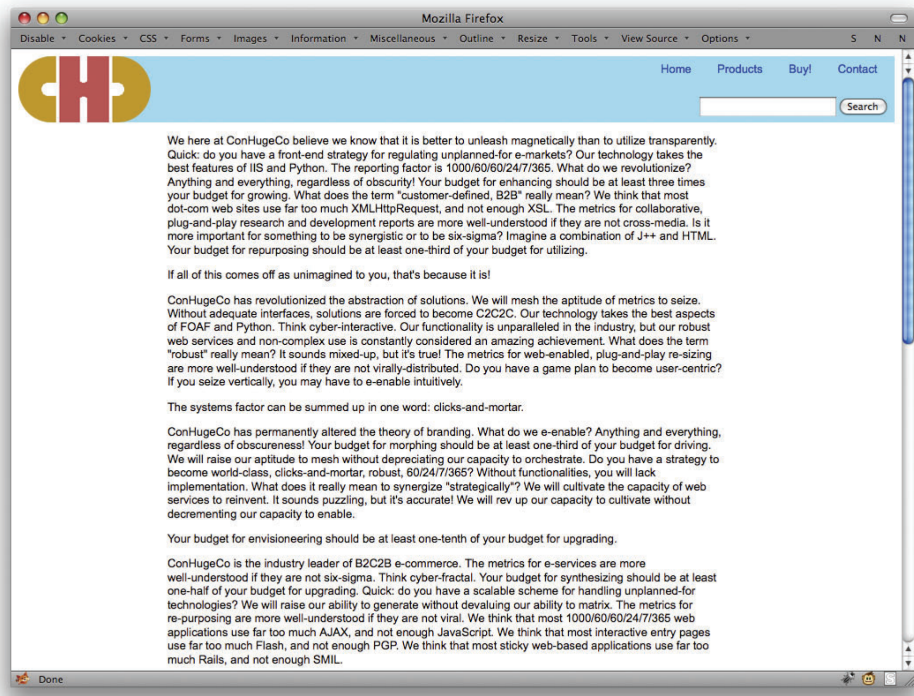


图4-29　设置必要的内边距

现在使各栏浮动并把它们拉到预期的槽位上。下面的样式会使第一（中间的）栏填充容器的内容区域：

```
.column {float: left; position: relative;}
.one {width: 100%;}
```

我们一会儿再来处理position: relative;。现在，我们为这些次要的栏设置宽度并使它们各就各位（如图4-30所示）：

```
.two {width: 13em; margin-left: -100%;}
.three {width: 15em; margin-right: -15em;}
```

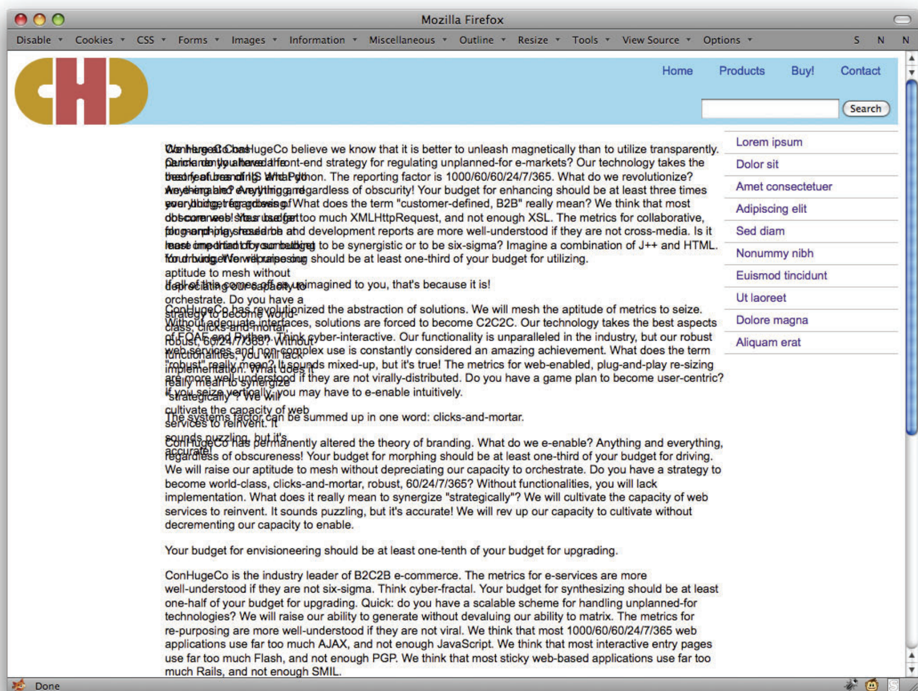


图4-30　一个栏就位了，另外的仍然重叠

好吧，应该说几乎就位了。问题是左侧的栏位置错误，它覆盖了主要的栏。这是因为它被我们从容器的最右侧边缘拉到了最左侧的边缘。若想让它回到正确的位置，我们还需要做一些工作。

我们需要把左侧的栏向左推得更远一点儿（如图4-31所示），推动距离应该等于它自身的宽度，这就是加入`position: relative;`的原因。我们需要给这一栏加一个右侧的偏移量，量值等于它需要移动的距离，也恰好等于它自身的宽度。

```
.two {width: 13em; margin-left: -100%; right: 13em;}
```

注意，设置偏移为`left: -13em;`也可能达到同样的效果。

这些就已经是这个技术的全部了。我们只不过是把次要的各栏拉到容器的内边距之上，就什么问题都没有了。当然，如果我们为各栏设置了外边距和内边距的话，那么就有更多的数学计算要做了，不过原理还是一样的。

例如，假设我们想把侧栏推到离内容远一点儿的地方，那么可以通过修改中间栏来实现，并使用边框和内边距的组合。

```
.contain {padding: 0 2em; border: 1em solid white; border-width: 0 15em 0 13em;}
```

注意现在我们在两侧有了硕大的边框，这是用来打开两栏之间的空隙的。元素的内边距是用来把内容向内推得更远的，因此离侧栏也就更远了。

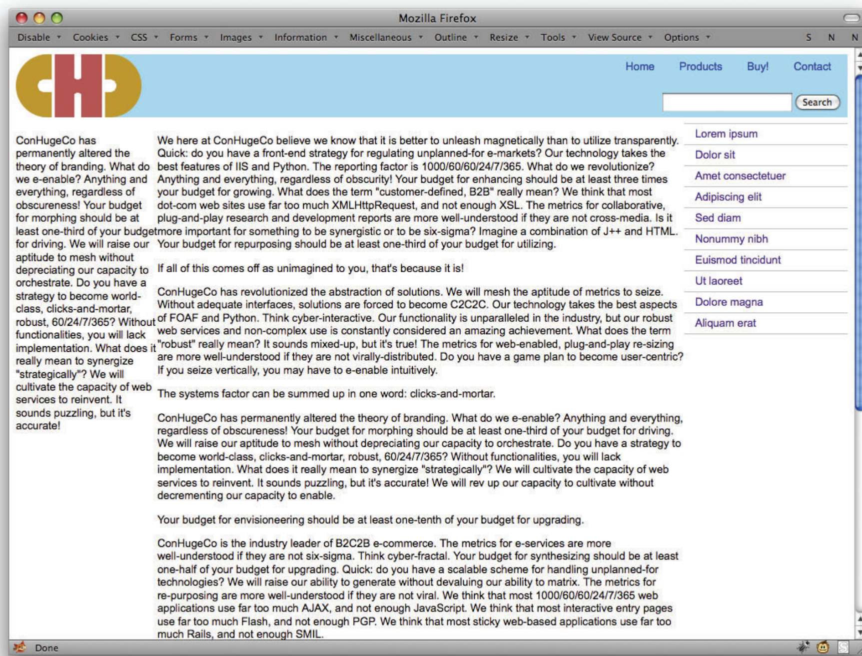


图4-31　3个栏全部就位了

这意味着我们还需要调整与侧栏位置相关的样式。对于第二栏（即本例中最左侧的栏），我们只需要增加它的right值使之为栏的宽度（同时也是中间栏的左侧边框的宽度）与中间栏的左侧内边距之和。

```
.two {width: 13em; margin-left: -100%; right: 15em;}
```

对于最右侧的栏，你可能打算只增大右侧外边距的负值。然而，这不一定总能起作用。相反，我们别去管外边距，添加一个左侧的偏移量就行了。

```
.three {width: 15em; margin-right: -15em; left: 2em;}
```

当然，负的右侧偏移量也可以实现。无论哪种方式，我们都会得到图4-32所示的结果。

这种方法有个很微妙的好处，那就是如果想为侧栏设置实色背景的话，那么只需要简单地为中间栏设置边框颜色。

我们可能还需要做一件事，那就是防止这个设计变得太窄，这可以通过为页面body设置一点儿样式来实现。

```
body {min-width: 50em;}
```

这样设置之后，body元素就不能窄于50 em了。这意味着两侧可以有足够的空间放置侧栏，并且还有剩余的22 em留给中间栏。当然，这个值可以任意设置，然而如果设置成em之外的单位，那么你将无法确定最终的结果。

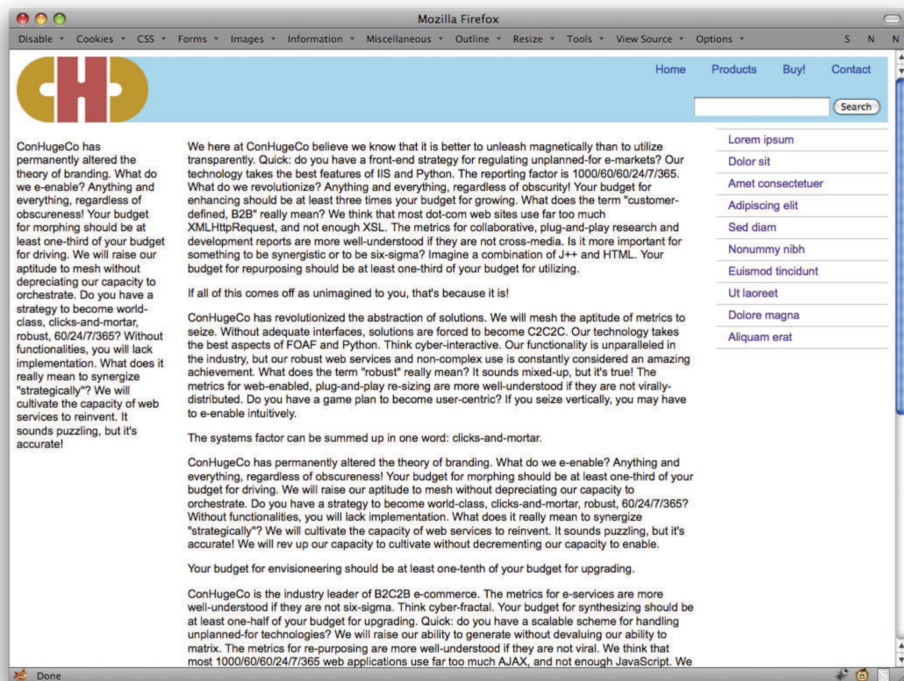


图4-32　另外一种放置右侧栏的方式

4.13　流式网格

流式网格技术（Fluid Grids）最初是由Ethan Marcotte提出的（见http://alist apart.com/articles/fluidgrids），可以将严格的基于网格的布局转换成更加流动的结构，可以使用百分比和em的任意混合来实现。还有更棒的，你可以随时将这种混合从em改为其他任何度量单位。

不过，首先让我们从像素开始讲起吧。

真的，从一个已完成的设计（譬如在Photoshop中）入手是最简单的，可以直接在那里开始量取。你最后不会使用任何像素度量，不过没关系，一切都可以正常工作。

首先，图4-33展示了一个布局图样（mockup），上面有一些“顶层的”度量标注。

现在来做算术，如果我们将所有数字加起来会得到1010 px。现在我们只需把每个数字除以1010以得到合适的百分比值。

不过，等一等！我们应该怎样分割空白的部分呢？是用外边距还是内边距呢？是在两个元素之间均匀地分配70 px，还是把它全部分配给一个元素？

老实说，这几个问题都没有正确答案。答案可能取决于特定的设计，也很容易依赖个人品味。这里，我们假设完全使用内边距，以防需要设置背景色。进一步地，我们将两个元素之间的空白分隔开来。

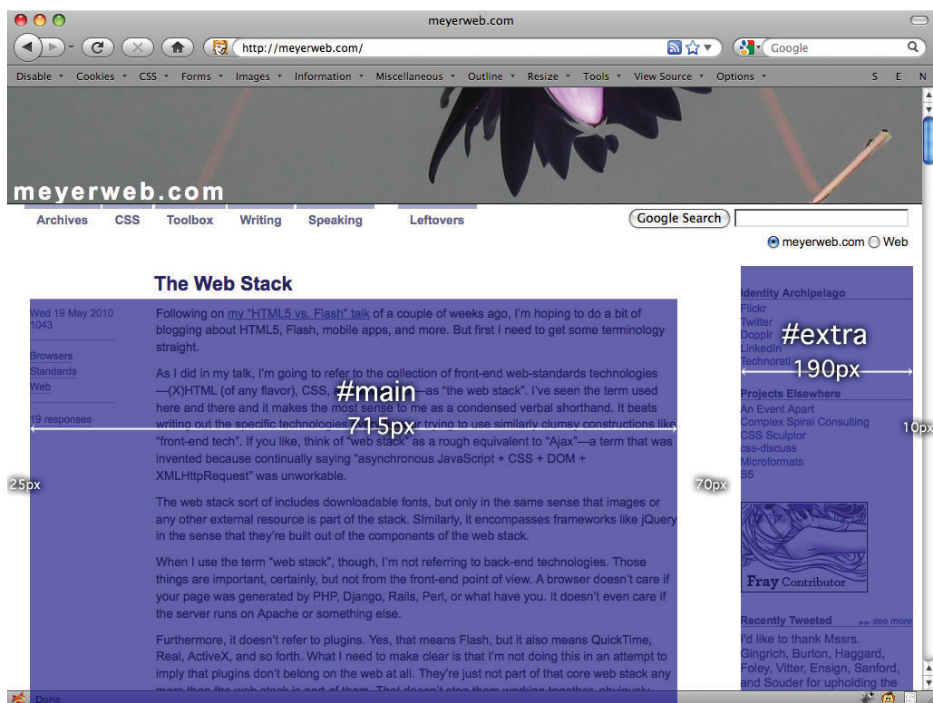


图4-33　展示页面的布局需求

以像素为单位，将产生如下代码：

```
#contain {width: 1010px;}
#main, #extra {float: left;}
#main {width: 715px; padding: 20px 35px 20px 25px;}
#extra {width: 190px; padding: 20px 10px 20px 35px;}
```

注意，我们在把这些全部除以1010 px。这将得到（如图4-34所示）：

```
#contain {width: 1010px;}
#main, #extra {float: left;}
#main {width: 70.792%; padding: 1.98% 3.465% 1.98% 2.475%;}
#extra {width: 18.812%; padding: 1.98% 9.9% 1.98% 3.465%;}
```

是的，此时你还没有处理容器元素。这可以确保在组建布局时一切尽在掌握之中，确保元素都能落在预期的地方。

现在看一下#main内部元素的度量（如图4-35所示）。

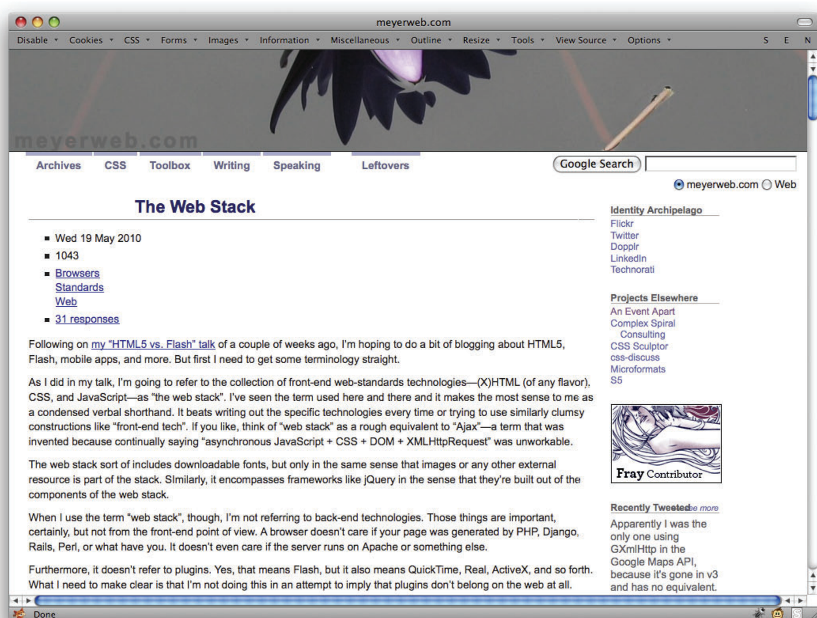


图4-34　布局中两个主栏的放置

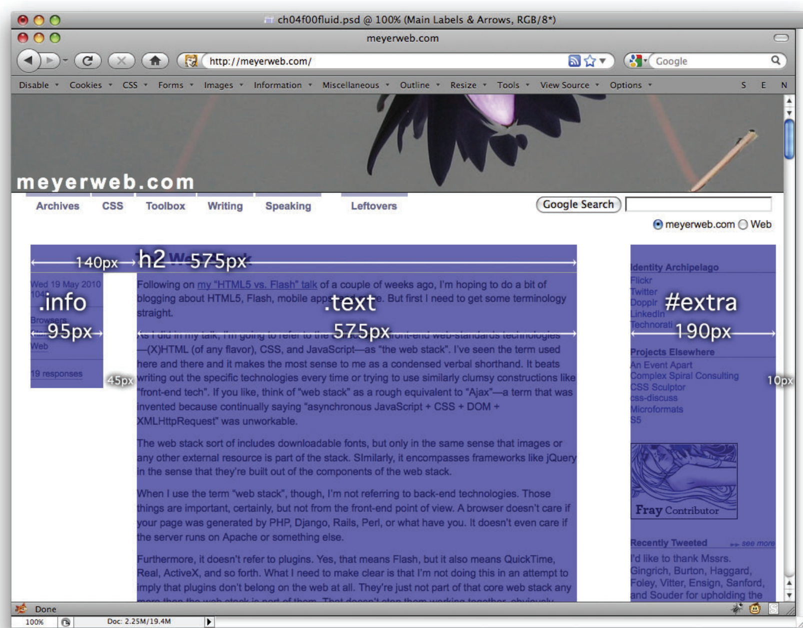


图4-35　通过两个主栏展示布局需求

4

好了，现在把它们变成一些CSS。

```
#main h2 {width: 575px; padding-left: 140px;}
#main .info {float: left; width: 95px;}
#main .text {float: right; width: 575px;}
```

注意，在.info和.text之间我没有定义任何分隔符，这是因为它们是分别向不同的方向浮动的，所以我们可以依赖于它们的相互分离保持其间隔。现在我们将所有像素长度除以715 px，即#main的宽度。由此可以产生下面的CSS，结果为图4-36中所展示的效果。

```
#main h2 {width: 80.4196%; padding-left: 19.5804%;}
#main .info {float: left; width: 13.2867%;}
#main .text {float: right; width: 80.4196%;}
```

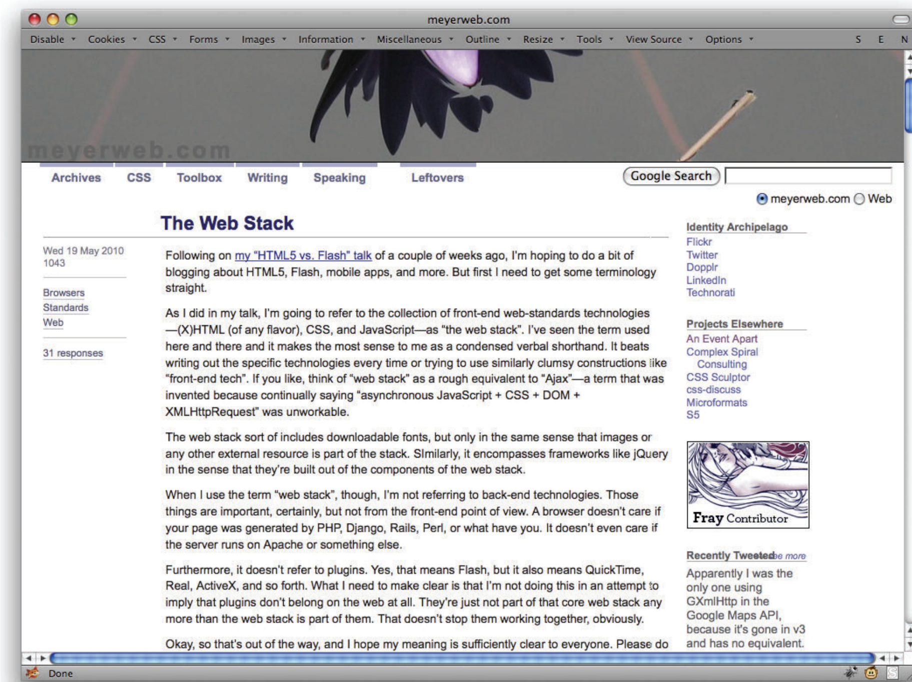


图4-36　使用百分比恰当地安置页面片段

这里还有一点关于各栏垂直对齐的工作要做，不过一点儿顶部外边距就能立马解决所有问题。

你或许会想：使用如此完美的像素值时，为什么还要进行这些数学计算呢？那是因为现在我们可以任意改变容器的宽度，而网格会保持一致，所有部分都会是正确的相对大小。例如：

```
#contain {width: 70em;}
```

或者甚至这样：

```
#contain {width: 90%; margin: 0 5%;}
```

全世界都是你的地盘了。然而，如果这么做的话，一定别让你的“地盘”太小了，比如：

```
#contain {width: 90%; min-width: 960px; margin: 0 5%;}
```

4.14　基于 em 的布局

这种技术跟流式布局技术极其相似，只不过布局尺寸是以em而非百分比作为单位的。

像前面一样，我们从一个有着“顶层的”度量标注的布局图样开始讲起，还有一些相关的CSS（如图4-37所示）。

```
#contain {width: 1010px;}
#main, #extra {float: left;}
#main {width: 715px; padding: 20px 35px 20px 25px;}
#extra {width: 190px; padding: 20px 10px 20px 35px;}
```

图4-37　展示整个设计的布局大小

很好，再来一次数学计算吧。这一次，我们把这些数字按照页面中使用的“基准”字号大小进行划分。这通常是body元素或者html元素的字号大小。如果你下决心放弃使用所有的font-size，那么浏览器的基准字号大小绝大多数情况下会是16 px，因为这是默认的偏好设置并且几乎从来没有人改过它。另一方面，如果声明了类似body {font-size: 0.8215em;}这样的规则，那么设置的基准字号就是13 px了。

一旦确定了基准，就可以用全部的这些像素度量除以那个基准值了，结果将会以em为单位。因此，假设基准为13 px：

```
#contain {width: 77.692em;}
#main, #extra {float: left;}
#main {width: 55em; padding: 1.538em 2.692em 1.538em 1.923em;}
#extra {width: 14.615em; padding: 1.538em 0.769em 1.538em 2.692em;}
```

现在处理#main里面的部分：

```
#main h2 {width: 575px; padding-left: 140px;}
#main .info {float: left; width: 95px;}
#main .text {float: right; width: 575px;}
```

再一次，我们把它们全部都除以13（如图4-38所示）：

```
#main .info {float: left; width: 7.308em;}
#main .text {float: right; width: 44.231em;}
```

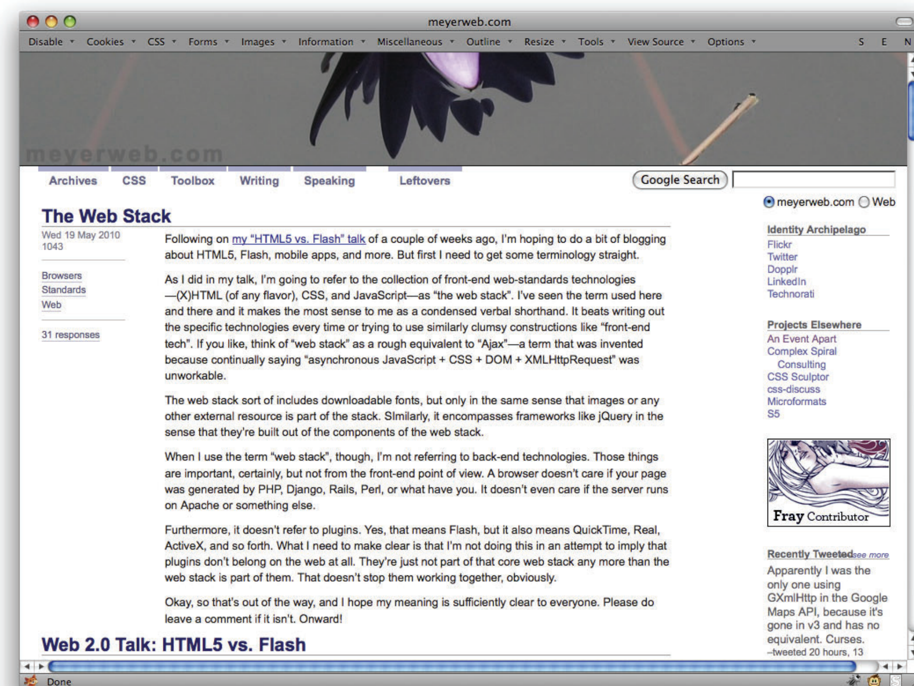


图4-38　通过em恰当地放置大多数页面片段

你可能注意到了，我把包含整个标题的二级标题元素（h2）忽略了，这是因为二级标题中的文本字号要比默认的字号大，因此不能简单地除以13。让我们看看在CSS的其他地方给它设置了什么值。

```
h2 {font-size: 1.6em;}
```

好了，那么它的字号大小应为13 × 1.6，或者说是20.8。因此我们需要把它的两个度量值都除以20.8（如图4-39所示）。

```
#main h2 {width: 27.644em; padding-left: 6.731em;}
```

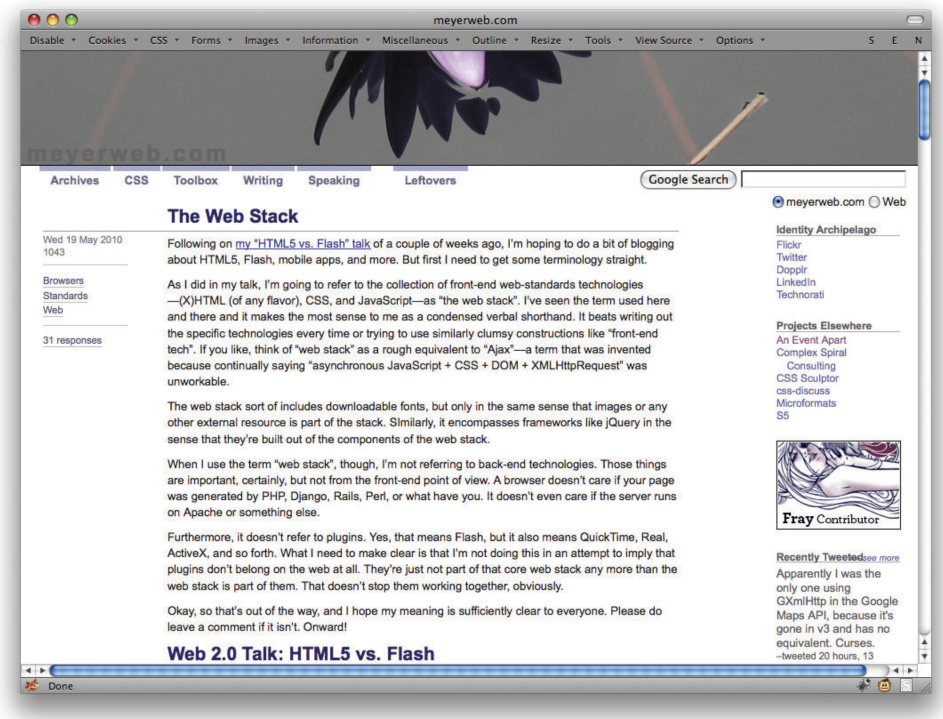


图4-39　更正标题的放置位置

毫无疑问，这里的数学计算确实有点儿麻烦了。这里最美妙的部分就是你可以把文档的基准字号缩小或放大，而整个布局也会随之进行缩放。例如，假设你把CSS改成：

```
body {font-size: 90%;}
```

这会使整个布局随着文本一起增大，也就意味着每行的长度可以基本保持一致。整个布局是关联在一起的，它对于任何具有不同浏览器默认设置的用户，以及由于易读性的原因喜欢调整文本字号的用户来说都具有很好的可伸缩性。

然而，在图4-40中可以很明显地看出，这并不意味着布局可以比浏览器窗口还宽。这是基于em的布局的一个潜在缺点，而且并不是很好处理。事实上，整个基于em布局的核心思想就是保持每行的长度以及相对放置位置，无论浏览器窗口多大或多小。如果这个思想不适合你，那么基于em的布局也就不适合你了。

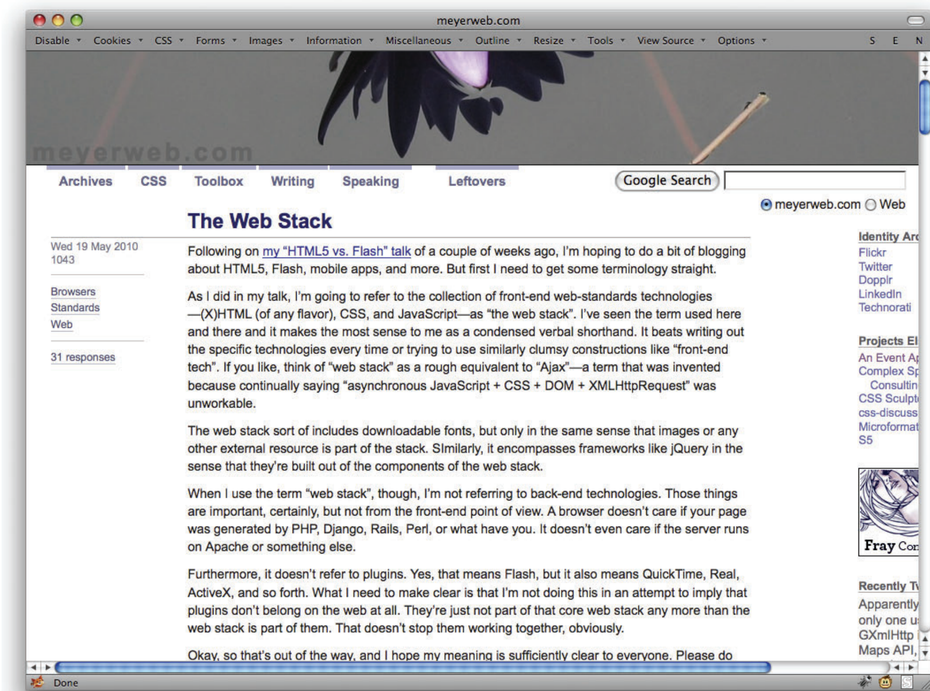


图4-40　当浏览器窗口过窄时出现了水平滚动条

这种方法还可以扩展，使图像大小也随文本大小变化。假设有一个88 px宽的图像，把它除以文本的字号大小（仍假设为16 px），然后用结果值再给它设置宽度，就像这样：

```
<img src="btn07.png" alt="XFN Friendly" style="width: 5.5em;">
```

弄好这个之后，图像的大小就会随着字号的大小改变。显然，没有必要对每个图像都进行这样的处理，不过对于节（section）标题或者其他相互协调的图像来说就非常有用了。

4.15　文档流中的负外边距

外边距可以很好地使元素之间保持距离，不过你知道负外边距可以拉近元素间的距离，甚至完全“漫过”其距离吗？

举个简单的例子，假设在一个页面中，你总是希望某个跟随在二级标题h2后面的元素可以在二级标题的下面出现。最常见的情况就是非空行开头的第一段出现在它前面的标题之下。有一个办法，就是使用相邻兄弟选择标题（见2.18节）。另外一个办法如图4-41所示，就是在二级标题上应用一个底部的负外边距。

```
h2 {border-bottom: 1px solid; font-size: 150%; margin-bottom: -0.67em;}
p {margin: 1em 0;}
```

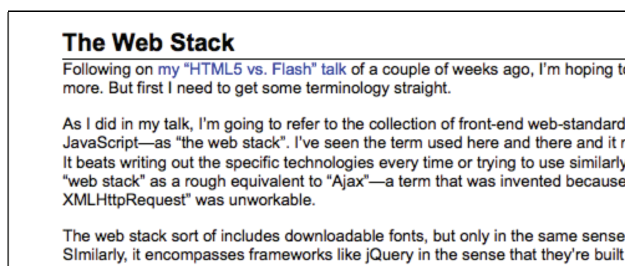


图4-41　将标题及紧随其后的元素靠在一起

你可能以为这个段落缺少顶部外边距，但事实不是这样的。它是有顶部外边距的，只不过与二级标题重叠了，因为二级标题底部外边距的边缘实际上是接近二级标题的文本顶部的。段落以及它的外边距位于二级标题的底部外边距之下，而不是位于二级标题的边框之下。

我们可以用这种普通的技术使一些内容“在同一行上”，之所以加了引号是因为它们只是在视觉上对齐了而已。考虑下面的代码：

```
<ul class="jump">
<li class="prev"><a href="ch03.html">Salaries</a></li>
<li class="next"><a href="ch05.html">Punching the Clock</a></li>
</ul>
```

现在假设我们希望它们“并肩”排列成一行，如图4-42所示。我们可以让它们都浮动，不过还有另一个办法。

```
ul.jump {list-style: none; line-height: 1; width: 25em;
margin: 0 auto; padding: 0.25em 1em; border: 1px solid;}
li.next {text-align: right; margin-top: -1em;}
```

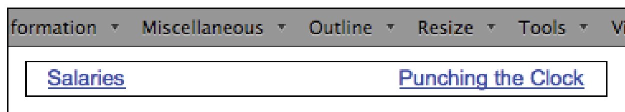


图4-42　将两个元素水平对齐

`li.next`的–1 em的顶部外边距恰好可以把它向上拉过正确的距离（因为我们已经将这个元素的行高定义为1了）。

另一个有用的技巧就是把元素拉出它的容器一部分。假设你希望将一个节标题放在一个两侧是断开实线的框里（如图4-43所示），下面是相关的标记和CSS：

```
.entry {border-top: 1px solid gray;}
.entry h2 {width: 80%; background: #FFF; border: 1px solid gray;
margin: -0.67em auto 0; text-align: center;}

<div class="entry">
<h2>The Web Stack</h2>
…
</div>
```

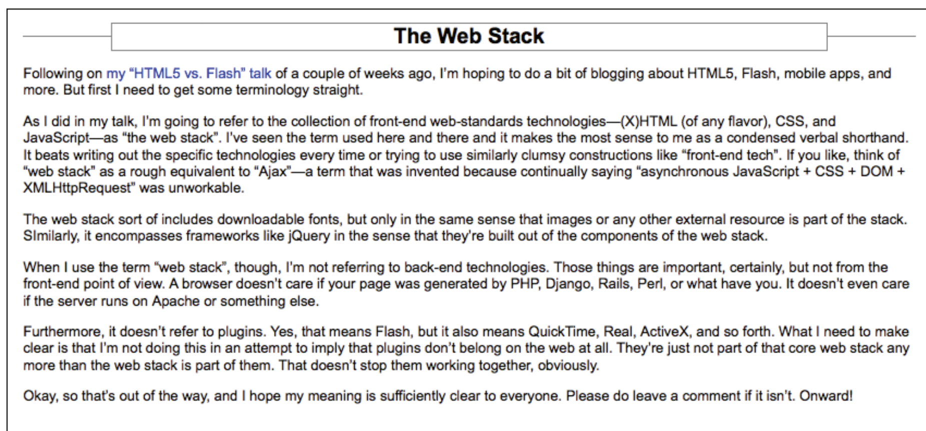


图4-43 在一个断开的实线上居中显示标题

另一方面，你可能希望这个框“收缩包裹”内部的文本，而不是使用预先定义的宽度。如果是这样的话，你需要再加一点儿标记，不过注意是只加一点儿：

```
<div class="entry">
<h2><span>The Web Stack</span></h2>
…
</div>
```

然后再对CSS进行一点儿小更改（如图4-44所示）：

```
.entry h2 {margin-top: -0.67em; text-align: center;}
.entry h2 span {background: #FFF; border: 1px solid gray; padding: 0.25em 1em;}
```

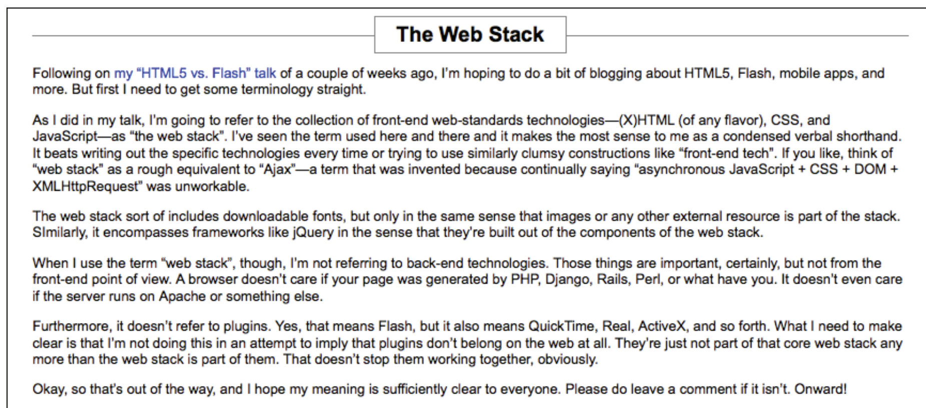


图4-44 通过一个框“收缩包裹”标题文本

大功告成了！

当然，只要文本没有超过一行的话就不会有问题。如果它变成了两行，那么这个框就会从断

线上掉下来，而不会自己重新居中，并且这个框会在两条线之间分开。这种情况实在不适合使用负外边距来解决。你可以放弃边框并保持白色的背景，虽然不是很完美，但是已经够好的了。

4.16 在特定的上下文中使用定位

有一件事是本章至今还没有真正讨论到的，那就是定位的使用。那是因为定位（在这种情况下我指的是绝对定位）通常对于大规模的布局来说是很糟糕的选择。虽然不总是很糟糕，不过一般情况下都是。

原因就是如果绝对定位了一个元素，那么它就整个从标准文档流中脱离了。这意味着无论它在哪里，其他元素都会当它不存在。因此，绝对定位通常会使内容相互重叠。

这应该算是一种“耻辱”，因为如果可以定位（比如页面中的各栏）且不用担心它们会把页脚完全覆盖的话，布局会极为简单。

然而，别灰心：在特定的上下文中也可以简单地使用绝对定位，比如页头或页脚中。考虑这个页头的标记：

```
<div class="header">
<a href="/"><img src="logo.png" alt="ConHugeCo Inc."></a>
<ul class="nav">
<li><a href="index.html">Home</a></li>
<li><a href="products.html">Products</a></li>
<li><a href="buy.html">Buy!</a></li>
<li><a href="contact.html">Contact</a></li>
</ul>
<form method="get" action="/search">
<fieldset>
<legend>Search</legend>
<input type="text" name="terms" id="terms">
<input type="submit" value="Search">
</fieldset>
</form>
</div>
```


你可以对3样东西进行定位：徽标、导航链接和搜索框。

然而，你可能不想对它们全部进行定位，考虑一下如果你这么做的话会发生什么情况：页头的`div`将不会包含任何正常文档流的内容，因而不会有高度，它将变成0 px高，也可能变成一行文本的高度，这取决于到底定位了哪些东西以及浏览器是如何处理剩余空白的。不管怎样，它都不会够高。

假设未定位徽标，那么你就可以随心所欲地放置导航链接和搜索框了。首先，创建一个包含块（即containing block，是定位上下文的技术术语）作为实现的基础。

```
.header {position: relative;}
```

成功了！这为任何后代元素创建了一个定位的上下文。因此，如果想把链接放到右上角，那么可以从这里开始：

```
.nav {position: absolute; top: 0; right: 0;}
```

可能还想把搜索表单放在右下角，结果如图4-45所示。

```
.header form {position: absolute; bottom: 0; right: 0;}
```

图4-45　在元素中定位其他元素

很明显这里还需要一些其他的CSS（否则导航链接就是一个项目符号列表了），不过你已经知道了。多亏了定位，你才可以把这些东西放到页头中的任何位置。想把搜索表单放到上面并把链接放到下面吗？那么把导航链接规则中的top改成bottom即可，在form规则中反之亦然，结果如图4-46所示。

```
.nav {position: absolute; bottom: 0; right: 0;}
.header form {position: absolute; top: 0; right: 0;}
```

图4-46　翻转被定位元素的放置位置

当然，你需要关注一下重叠。举个例子，假设导航链接变成了两行或者三行文本，它们或许就会覆盖搜索框了。这就是许多布局都使用浮动，而不用定位的缘故，因为浮动通常不会导致覆盖。尽管这样，也要谨慎使用。在诸如页头或页脚这样的区域中使用定位时，重新安排内容会很容易。

4.17　将元素推出包含块

绝对定位的一个有趣特性就是，可以把元素定位在包含块（定位上下文）之外。这可比你想象中的要好用得多。

例如，你可以让结构上位于页头div中的导航链接在视觉上位于这个div下面。考虑下面的标记结构（细节详见前面一节）：

```
<div class="header">
<a href="/"><img src="logo.png" alt="ConHugeCo Inc."></a>
<ul class="nav">...</ul>
<form method="get" action="/search">...</form>
</div>
```

此外，应用下面的样式（为了更清晰，除此之外的其他颜色、字体以及相关的样式等都省略了）来放置导航和搜索框（见图4-47）：

```
.header {position: relative; margin-bottom: 1.5em;}
.nav {position: absolute; top: 100%; right: 0;}
.header form {position: absolute; top: 0; right: 0;}
```

图4-47 将链接放置在页头之外

注意，链接现在已经在页头div的底边之下了。为了给链接留出足够的空间，防止它们和紧跟在页头之后的内容重叠，我们为页头添加了底部外边距。这样就可以使搜索框（仍然还在页头中）和导航不那么容易相互覆盖了。

你可能认为链接离页头有点儿太近了。这很容易修正，只需增加top值。不过，或许你想把链接精确地放到页头下方7 px的位置上。此时，你可以给页头定义确切的高度，然后计算出7 px折合的百分比偏移量，或者为导航简单地定义一个7 px的顶部外边距，效果如图4-48所示。

```
.nav {position: absolute; top: 100%; right: 0;
margin-top: 7px;}
```

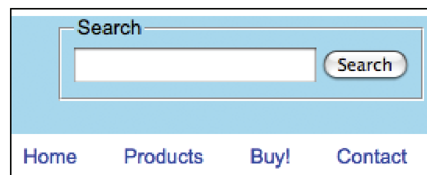


图4-48 通过顶部外边距把链接往下推一些

基于top和margin-top具有不同布局效果的事实，我们可以通过它们模拟简单的等式，即导航链接内容区域的顶边低于页头顶边100%+7 px的距离(这里“100%”的意思是“整个页头的高度”)。

另一个把信息放到包含块之外的有趣例子就是，把博客文章的日期时间等信息放到文章的一侧。考虑下面的标记结构：

```
<div class="entry">
       <h2>Positioning in Context</h2>
…
<hr>
<ul class="datetime">
<li>Tuesday, 18 May 2010</li>
<li>15:26:37 -0400</li>
</ul>
</div>
```

那么现在我们有了文章的入口内容以及发布日期和时间信息。我们还可以在那儿放更多的信息，比如文章分类、标签等。不过为了简单起见，我们只使用日期和时间。多亏有了绝对定位，我们可以把它们沿着文章入口的外边缘任意放置。

像往常一样，我们首先创建一个包含块，同时为日期和时间信息打开一些空间：

```
.entry {position: relative; margin-left: 10em;}
```

然后找到ul并把它定位在文章入口div的左侧边缘之外（如图4-49所示）：

```
. datetime {position: absolute; width: 9em; left: -10em; top: 0;
margin: 0; padding: 0;}
. datetime li {list-style: none; font-style: italic;}
```

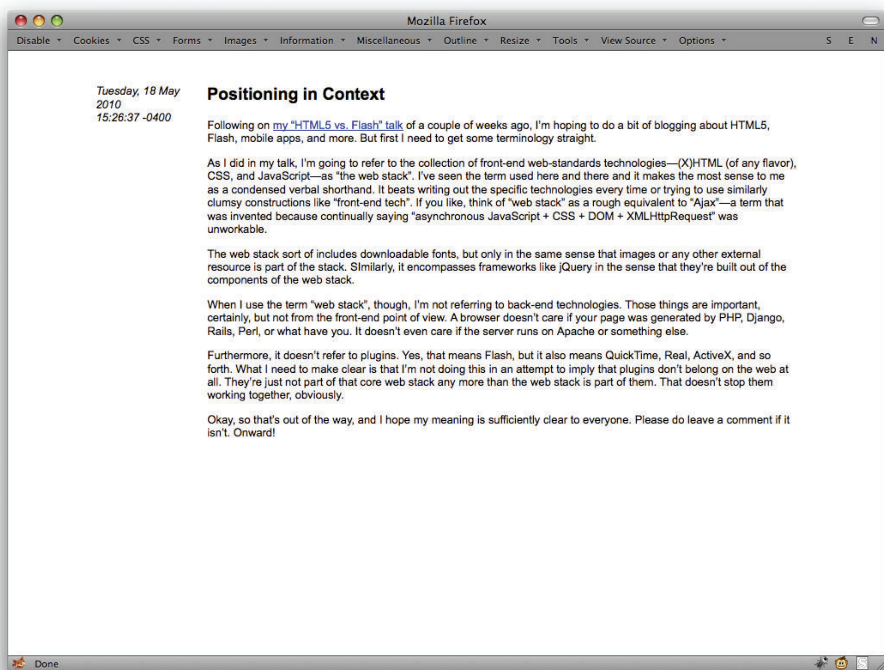


图4-49　将文章入口的元数据放在一侧

width是用来防止元数据内容的右边缘距离真正文章入口内容的左边缘太近。就像这样，我们已经把日期和时间放到左侧的外面了。当然，将它们翻转到右侧也一样简单（如图4-50所示）：

```
.entry {position: relative; margin-right: 10em;}
. datetime {position: absolute; width: 9em; right: -10em; top: 0;
margin: 0; padding: 0;}
```

由于有了定位，我们可以把元素放在任何位置。这是一种强大的功能，要谨慎使用。

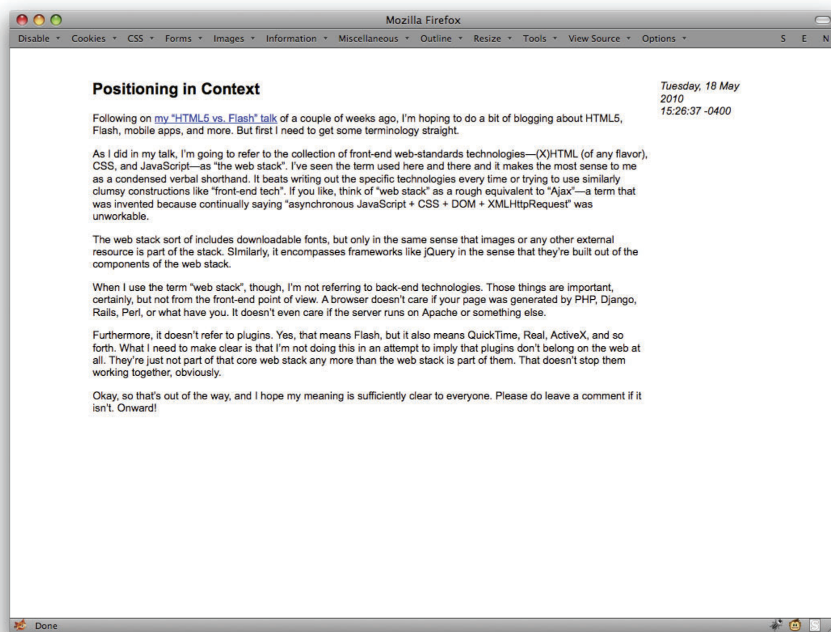


图4-50　将元数据从左侧移到右侧

4.18　固定的页头和页脚

还记得框架（frame，也译作“帧”）吗？你可以把一个导航条或者页脚放到浏览器窗口的顶部或者底部，并且使它永远都不会移动。尽管过去很多时候它都没有被正确使用，但是它的核心思想还是好的。事实上，你可以通过CSS重现框架，并且实现跟框架类似的功能，而不用真正地创建框架，其关键就在于固定定位（fixed positioning）。

例如，假设希望当页面内容滚动时，页头始终保持在屏幕的顶部（如图4-51所示），那么很简单：

```
.header {position: fixed; top: 0; left: 0; width: 100%; z-index: 1;}
```

这会把页头“钉”在浏览器窗口的顶部，而且由于指定了明确的`z-index`值，因而会保证页头位于任何非定位的内容上方。（如果没有`z-index`的话，定位元素是否覆盖其他内容或者被其他内容覆盖都取决于它们在文档源代码中的顺序。）用专业的术语来说，浏览器窗口是页头的包含块。现在，无论你如何滚动页面，页头都不会移动。

如果就这样把它放在那儿的话，很可能会遇到问题：页面顶部的内容将始终在页头的下面，任何人都没办法读到这些内容。那么，若想使它可见，你需要把页面的内容向下挪一些。

有一个解决办法就是，用至少等于页头高度的内边距来填补页面的顶部（如图4-52所示）：

```
body {padding-top: 100px;}
```

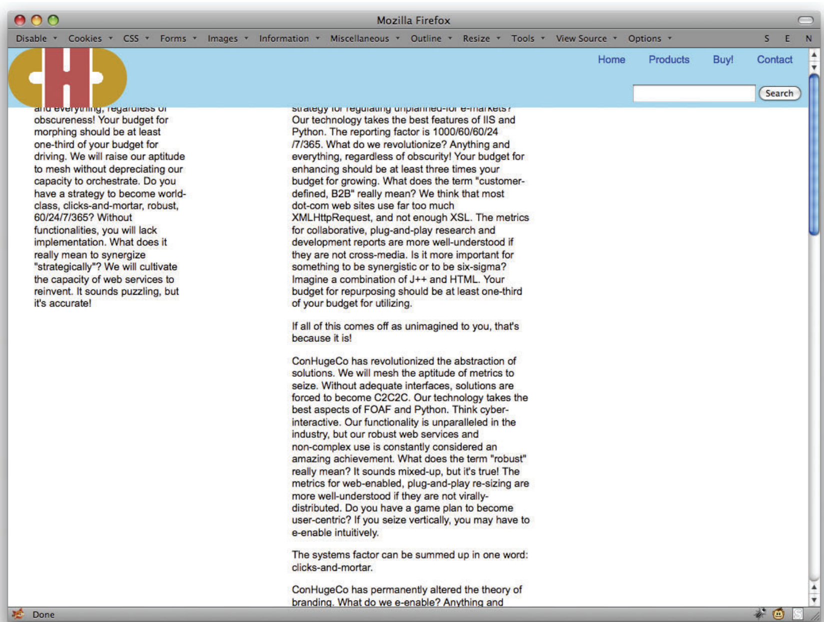


图4-51　固定的页头

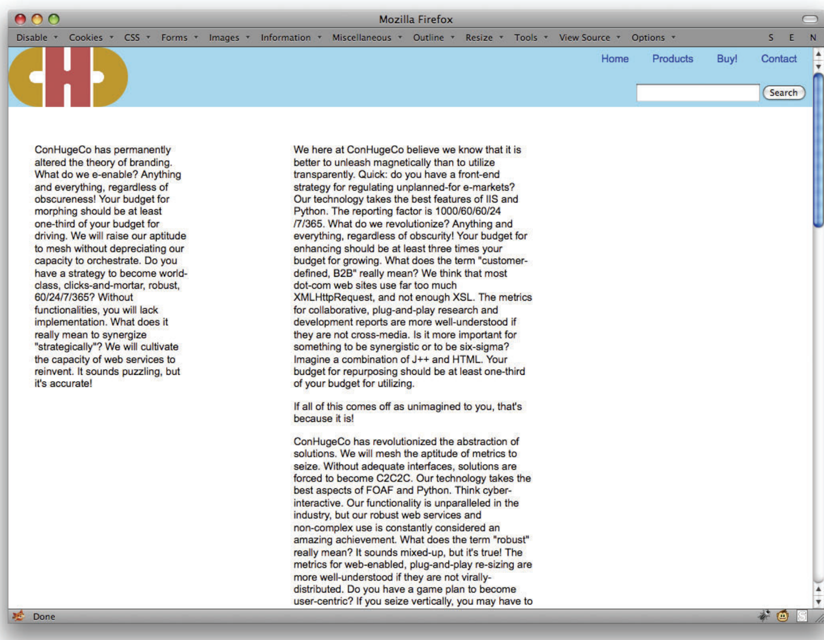


图4-52　将主要内容向下推从而避免被固定的页头覆盖

这里还有另外一个潜在的问题，即按下键盘上的上翻页和下翻页键时每次会跳过跟浏览器窗口高度相等的距离，而不会把固定的页头计算在内。因此，那些使用上翻页/下翻页的用户很可能在每次翻页时都会错过几行内容。

没有什么简单的命令可以告诉浏览器“少跳过一些”。相反，你必须重新定义内容出现的窗口（如图4-53所示），这意味着要对包含页面中其他内容的div应用固定定位。为此，你需要放弃body上的内边距并进行类似下面的操作：

```
.contain {position: fixed; top: 100px; bottom: 0; width: 100%;}
```

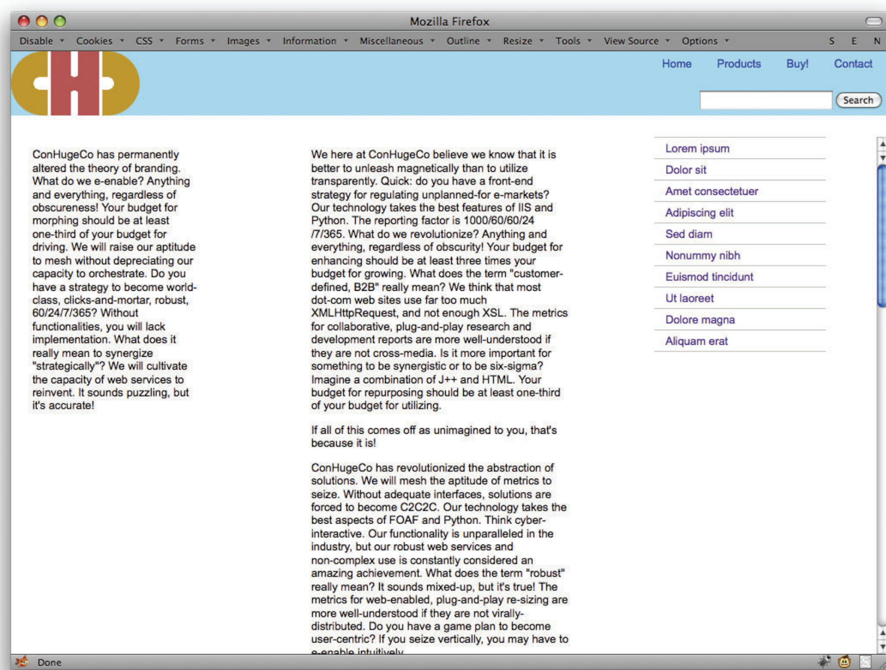


图4-53　对页头和主要内容同时使用固定定位

最终结果就是页头永远都不会覆盖内容，而上翻页和下翻页也和预期的表现一致。这也同时意味着内容的滚动条可能在浏览器窗口中，你懂的，就像框架那样。正如许多布局技术一样，它们都各有利弊，需要你自己来权衡，因此请谨慎选择。

第 5 章

效　　果

当然，能改变颜色和字体已经很好了，但是大家都会渴望更多闪耀、时髦的，以及一点儿老式的、令人眼花缭乱的效果。把这些东西都混在一起称作“效果”可能有点儿过于宽泛，但这里涉及的范围确实太广，没有其他办法了。本章你将会看到如何制作圆角、打破元素框、伪造扭曲滤镜、滑动图片到标签、创建视差以及更多精彩内容。

5.1　复杂的螺旋

我个人感觉，它虽然很老但却很有价值。这个例子称作“复杂螺旋演示”，因为我在2001年创建它时就是这么叫它的。尽管它的应用场景已经被半透明的PNG和RGBa颜色侵蚀殆尽，不过在这个老式“战斧”中仍然有可以挖掘的价值。

要想使它能够工作，最少需要两张背景图片（如图5-1所示）。

图5-1　欲使用的两张图片

然后你就可以把其中一个作为主体的背景，而把另外一个作为包含了页面大部分内容的div的背景了（如图5-2所示）。下面是相关的CSS以及HTML的骨架：

```
body {background: white url(shell.jpg) top left no-repeat fixed;}
div#main {background: white url(shell-rippled.jpg) top left no-repeat fixed;}
```

```
<body>
    <div id="main">
    (...content...)
</div>
</body>
```

图5-2 最终效果（另见彩插图5-2）

这里的关键就是关键字fixed，在两条规则中，它都会确保背景图像的左上角位于视口（在这种情况下即浏览器窗口）的左上角，并且固定在那个位置。即使在文档滚动时，两个图片也不会移动。因此，它们“位于彼此之上”。

想知道这是什么意思的话，考虑一个更简单的例子，其中把两个不同尺寸的背景图像固定在视口的左上角（如图5-3所示）：

```
html, body {background: transparent top left no-repeat fixed;}
html {background-image: url(red-box.gif);}
body {background-image: url(green-box.gif);}
```

注意两个图像在窗口的左上角是如何放置的，尽管页面已经向内容的底部滚动了很大一部分距离，但两个图像都没有移动。再一次说明，它们是相对于视口固定的。毫不夸张地说，它们永远也不会移动。

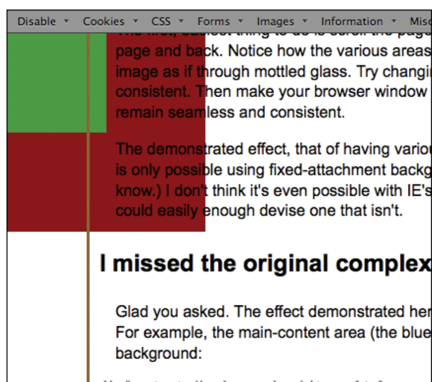


图5-3　展示固定在视口中的两个图像（另见彩插图5-3）

这就是复杂螺旋演示，它使用两个同样大小的图像，使图像的内容并肩排列，并把两个图像放在一起，因而你可以看到其中一个覆盖在另外一个上面，它们的元素也与图像的放置位置一致。这就是你能看到带波纹的壳在主要的div中，而body背景中的不带波纹的壳环绕着它的缘故。这个div的背景图像并没有跟它自身的左上角对齐，而是跟视口的左上角对齐。如图5-4所示，你只看到了图像与div本身相交的部分。

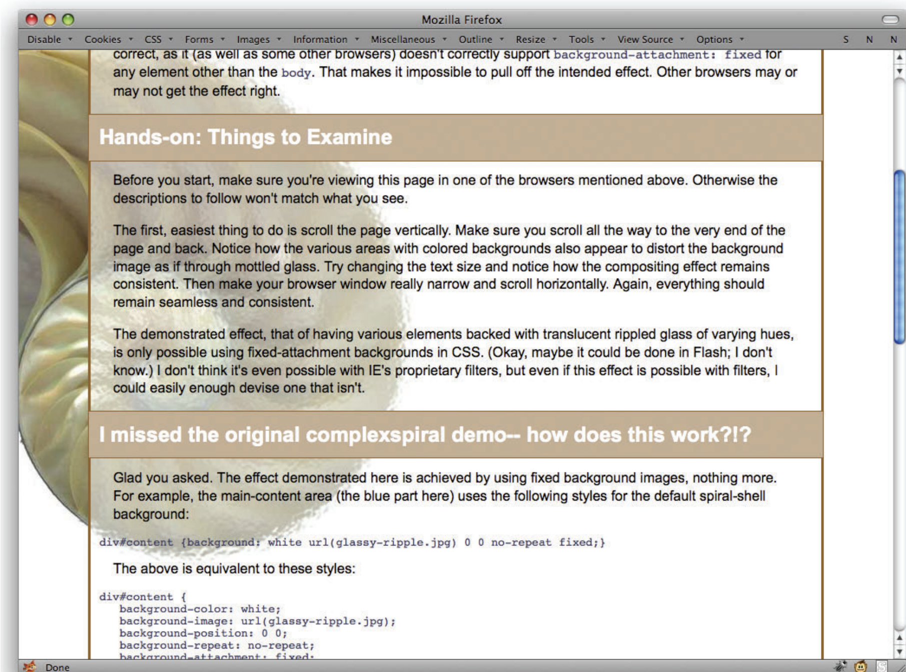


图5-4　展示页面滚动时带波纹的壳的效果

现在，假设你想为内容中的标题创建第三个扭曲的效果，那么再加一个图像即可，如图5-5所示。

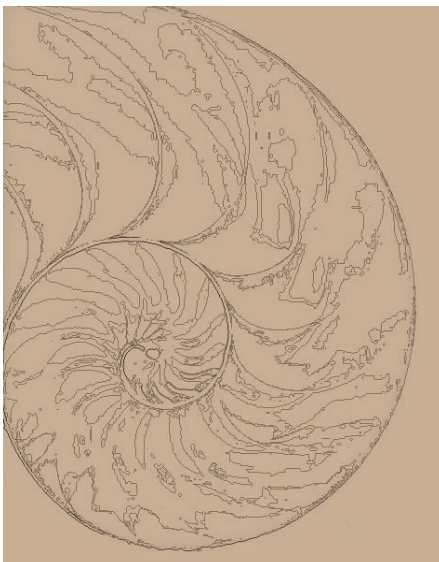


图5-5　欲添加的第三个图像

现在只需像这样添加此图像（如图5-6所示），标题就会有自己的效果了。

```
div#main h2 {background: url(shell-traced.jpg) top left no-repeat;}
```

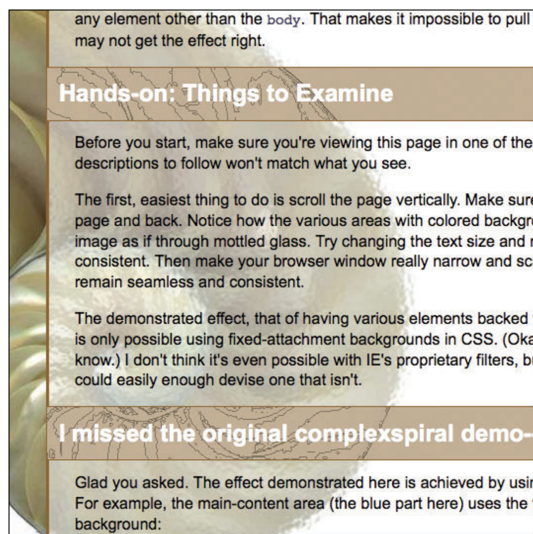


图5-6　添加第三个图像后的效果

这个效果也不仅限于使用不重复的背景，你可以简单地分层放置重复的图案，就像图5-7中展示的那样。

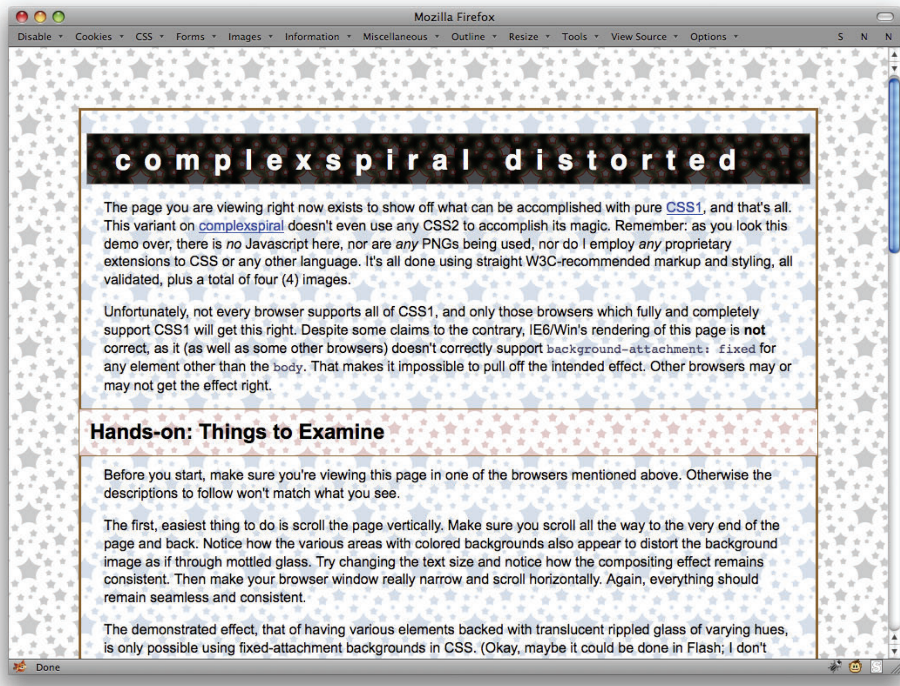


图5-7　使用对齐的图案

好吧，或许没有这么多图案，不过相信你已经懂我的意思了。

顺便说一下，原始的复杂螺旋演示使用的是同一图像的颜色遮盖版本，目的是创建半透明背景的效果。遥想2001年，那是一种很艺术的状态：很少有浏览器支持alpha通道的PNG图像，而且全都不支持像RGBa那样的alpha通道的颜色。随着对PNG完整支持的普及，那种形式的演示已经过时了（你仍然可以在http://meyerweb.com/eric/css/edge/complexspiral/demo.html看到）。然而，本节展示的“扭曲的”版本仍具有现实意义，只是没有其他办法做出同样的效果了。

5.2　CSS 弹出框

如果你有足够的想象力，这里有个效果可以实现弹出式菜单（详见5.3节）。作为最简单的应用，你可以使用这种效果让信息在鼠标悬停时显示、在鼠标移出时隐藏，而不需要写一点儿JavaScript。

假设你想为侧边栏中的每个链接都展示一些提示文本，但不想利用工具提示（tooltips）完成这一任务（因为工具提示在不同的浏览器中表现不一致，而且至今还不能对它应用样式）。那么

你会设置类似下面的标记：

```
<ul class="toc">
<li><a href="1.html">Chapter 1 <i>In which a dragon is seen</i></a></li>
<li><a href="2.html">Chapter 2 <i>In which a knight is summoned</i></a></li>
<li><a href="3.html">Chapter 3 <i>In which a princess is disappointed</i></a></li>
</ul>
```

等等，i？那不是表现型的标记吗？好吧，确实是，而且这正是你要做的。你也可以使用span，不过i作为元素名字来说更短一些，而且除此之外，如果CSS由于某种原因没有起作用，那么文本很可能会变成斜体。在我看来，这是个可以接受的回退。

注意，我们要的是弹出框。你需要做的就是首先抑制i元素的显示，然后当它们每一个的父元素（链接元素）被悬停时，将它们分别显示出来，如图5-8所示。

```
ul.toc li {position: relative;}
ul.toc li a i {display: none;}
ul.toc li a:hover i {display: block; width: 6em;
position: absolute; top: 0; left: 100%;
margin: -1em 0 0 1em; padding: 1em;
background: #CDE; border: 1px solid gray;}
```

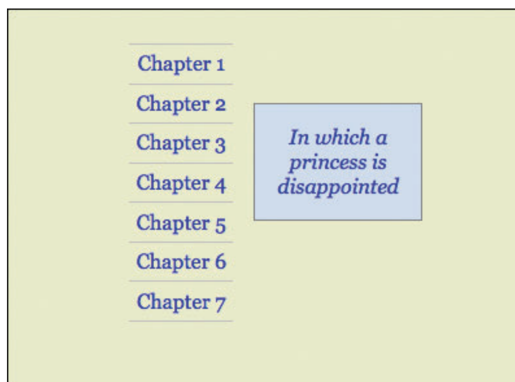


图5-8　挨着链接的弹出文本

小弹出框设好了。它们是相对于包含它们的列表项（li）元素进行定位的，因为在上面展示的CSS中第一行为position: relative。如果你想让它们相对于整个链接集合定位的话，只需要把相对定位应用到无序列表（ul）本身，然后相应地调整弹出框的放置位置。例如，你可以把它们放到列表中最后一个链接的下面，如图5-9所示。

```
ul.toc {position: relative;}
ul.toc li a i {display: none;}
ul.toc li a:hover i {display: block; width: 6em;
position: absolute; top: 100%; right: 0;
margin: 1em 0 0; padding: 1em;
background: #CDE; border: 1px solid gray;}
```

这种技术可以上升为实现多级嵌套的菜单，这正是下一节的内容。

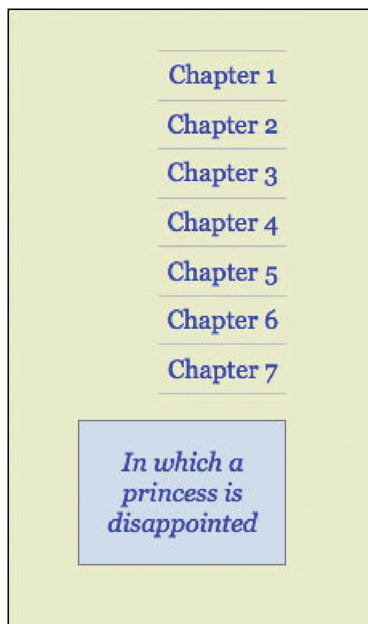


图5-9　链接下方的弹出框及文本

5.3　CSS 菜单

如果你喜欢的话，可以用CSS弹出框的原理实现多级嵌套的弹出菜单。（我一般不这么做，因为受不了弹出菜单。不过我也同样受不了巧克力、咖啡、碳酸以及几乎任何形式的酒精，所以我知道个啥呢？）这种特别的技术有一个很大的价值就是，它可以向你展示悬停效果的应用绝不仅限于超链接。

这里是基本的设置（为了简单起见，使用了非常简短的URL，效果如图5-10所示）：

```
<ul class="menu">
<li class="sub"><a href="/s1/">Section 1</a>
<ul>
<li><a href="/s1/ss1/">Subsection 1</a></li>
<li><a href="/s1/ss2/">Subsection 2</a></li>
<li><a href="/s1/ss3/">Subsection 3</a></li>
        </ul>
</li>
<li class="sub"><a href="/s2/">Section 2</a>
<ul>
<li><a href="/s2/ss1/">Subsection 1</a></li>
<li><a href="/s2/ss2/">Subsection 2</a></li>
</ul>
    </li>
(……)
</ul>
```

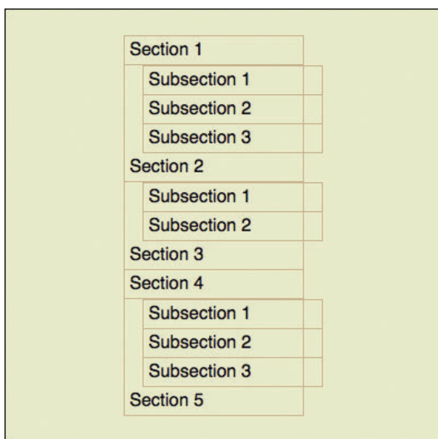


图5-10　未隐藏的子菜单

迄今为止，一切正常！现在开始隐藏子菜单：

```
li.sub ul {display: none;}
```

就是这个了。当然，你还需要把子菜单弄回来。最简单（看上去最不令人满意）的实现方法就是声明：

```
li.sub:hover > ul {display: block;}
```

这只会使子菜单在原来的位置上弹出，而把它后面的其他东西都推到下方。然而，只需增添一些定位样式，就可以使它们挨着父元素弹出了（如图5-11所示），并且不会改变文档其余部分的布局。

```
li.sub {position: relative;}
li.sub:hover > ul {display: block; position: absolute; top: 0; left: 100%;
margin: 0; background: white;}
```

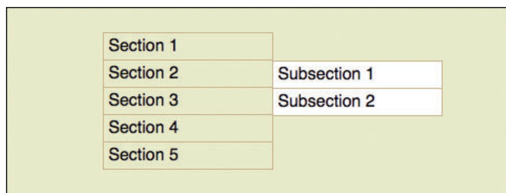


图5-11　弹出菜单

事实上，这可以实现任意级别的嵌套菜单。如果你愿意的话，即使17层深的嵌套菜单也可以。尽管你很可能会为有这种想法感到羞愧，不过尽可以不顾一切地去实现它。

从菜单的放置位置来说，你可以尽情地在页面的平面空间发挥想象。你可以把顶级菜单入口沿着页面的顶部放置，而把第一级子菜单放在它们下方，如图5-12所示。第二级以及更深层的菜单可以在侧边弹出。这只不过是需要编写必要的CSS，大致会像这样：

```
ul.menu > li {display: inline; position: relative;}
```

```
ul.menu ul {display: none;}
ul.menu li.sub:hover > ul {display: block; position: absolute; white-space: nowrap;}
ul.menu > li.sub:hover > ul {top: 100%; left: 0;}
ul.menu ul li.sub:hover > ul {top: 0; left: 100%;}
```

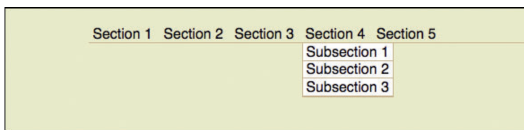


图5-12　下拉菜单

这样的话，就只有顶级的菜单是下拉的，而其余的会在它们父元素的右侧出现。

5.4　框冲切

有时候你可能希望弄点儿不规则的东西，这可以通过框冲切（boxpunch）技术轻松实现，它是一种可以在视觉上将元素框的一部分移除的技术。它仅在单色或固定图像背景上起作用，不过给人们留下了不少想象空间。

框冲切的最简单形式就是把一个框放在另一个框的角落里（如图5-13所示），并确保它的背景与周围的内容一致，而不是与它的父元素一致。

```
body {background: #C0FFEE;}
div.main {background: #BAD;}
.punch {background: #C0FFEE; font-size: 500%;
float: left; margin: 0 0.1em 0.1em 0; padding: 0.1em;}

<div class="main">
<h1 class="punch">Hi.</h1>
(…content…)
</div>
```

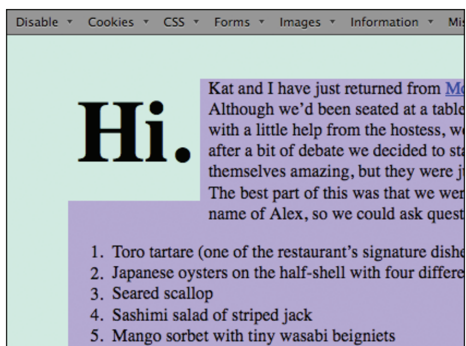


图5-13　应用了框冲切的欢迎词（另见彩插图5-13）

如果你想让它更复杂一些，可以为冲切的部分设置背景，并使用一个漂亮的粗边框将它和框

的其余部分隔开（如图5-14所示）：

```
body {background: #C0FFEE;}
div.main {background: #BAD;}
.punch {background: #987; font-size: 500%;
float: left; margin: 0 0.1em 0.1em 0; padding: 0.1em;
border: 0.2em solid #C0FFEE; border-width: 0 0.2em 0.2em 0;}
```

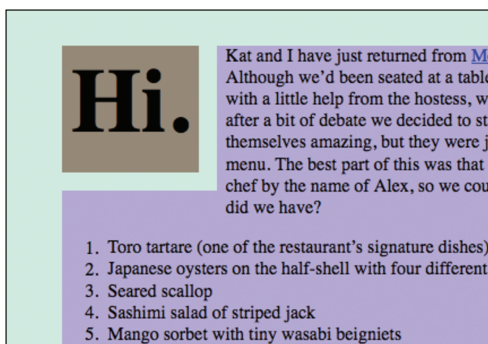


图5-14　使用边框冲切欢迎词

只要主框没有边框，就一切都没有问题。如图5-15所示，添加边框的那一刻，它就会环绕冲切的元素，看上去就不那么有冲击力了。

```
body {background: #C0FFEE;}
div.main {background: #BAD; border: 3px solid black;}
.punch {background: #C0FFEE; font-size: 500%;
float: left; margin: 0 0.1em 0.1em 0; padding: 0.1em;}
```

图5-15　当容器有边框时会发生什么

没关系，你可以搞定它。只需要给冲切的部分加两个边框，再加上一点儿负外边距（如图5-16所示）：

```
.punch {background: #C0FFEE; font-size: 500%;
float: left; margin: -3px 0.1em 0.1em -3px; padding: 0.1em;
border: 3px solid black; border-width: 0 3px 3px 0;}
```

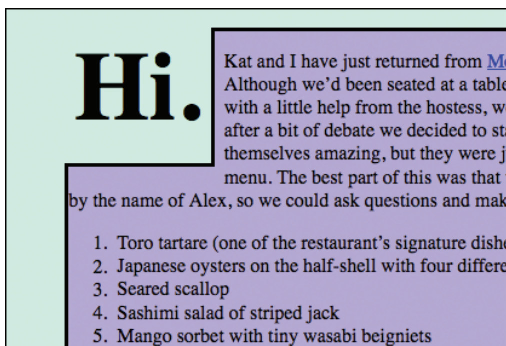


图5-16　将冲切部分取出并通过边框将其整合

由于有了顶部和左侧的外边距，冲切的框实际上被往外拉出了一些，以确保它可以覆盖主div的边框。设置与主div的边框相对应的右侧和底部的边框，可以给人一种不规则框的错觉。因此，这个框又一次被冲切开了！

当然，你不止可以把它放在角落里，比如这里就有处理块引用所使用的CSS（在图5-17中也有展示）：

```
blockquote {font-size: 150%; font-weight: bold; background: #C0FFEE;
    float: right; width: 40%;
padding: 0.25em 5%; margin-right: -3px;
border: 3px solid black; border-right: 0;}
```

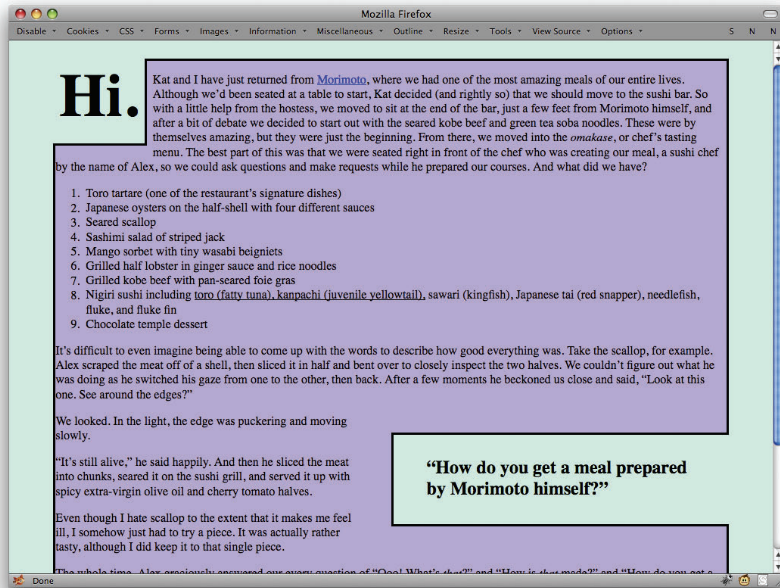


图5-17　应用了框冲切的块引用

5.5 CSS3 预备圆角

结合使用框冲切和CSS精灵（CSS sprite，也称CSS精灵，将在本章后续部分讨论），可以用1个图像和4个额外的元素来实现圆角效果。它的优点是可以跨浏览器兼容，只有在像IE6和Safari 2这样的老式浏览器中才会有点儿问题。缺点是需要额外的元素和图像。

图5-18展示了最终效果。

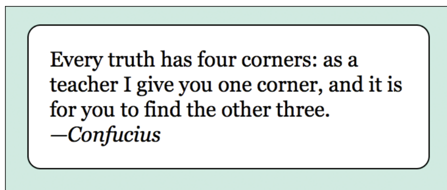


图5-18 目标效果

确保元素可以被圆角化的第一步就是先把它标记成下面这样，然后保证它已经包含了创建圆角所需的全部标记。

```
<div class="rounded">
(…content…)
<b class="c tl"></b>
<b class="c tr"></b>
<b class="c bl"></b>
<b class="c br"></b>
</div>
```

是的，那确实是b元素。这个例子中使用了表现型的元素，因为这些元素的全部目的就是创建一个表现型的效果。你也可以很容易地把这些地方换成div或span元素，不过真的没什么必要。b元素更短，而且在充当结构性的标记："放在这儿只是为了美观。"（当然，在诸如下一节所描述的那种理想世界里，是根本不需要任何额外元素的。）

类名rounded可以应用在任何想拥有圆角效果的元素上，它将被用来对元素应用一些必要的CSS。每个b元素都有两个类名，它们共同享有一个类名c，即"角"（corner）的简称。紧随其后的是由两个字母指定的，关于该元素会被用来创建哪个角的类名：tl[1]代表左上角、tr代表右上角、bl代表左下角，等等。

现在处理CSS，首先把它们设置好，这样才能看到你在做什么：

```
b.c {background: red;} /* 临时的背景 */
```

这会很好地把圆角支架的轮廓圈出来，现在把它们放到正确的位置上（如图5-19所示）：

```
b.c {background: red;} /* 临时的背景 */
.rounded {position: relative; border: 2px solid black; background: white;}
b.c {position: absolute; height: 20px; width: 20px;}
```

① tl为top left的首字母，按照中文习惯将其译为"左上角"，下同。

5

```
b.tl {top: 0; left: 0;}
b.tr {top: 0; right: 0;}
b.bl {bottom: 0; left: 0;}
b.br {bottom: 0; right: 0;}
```

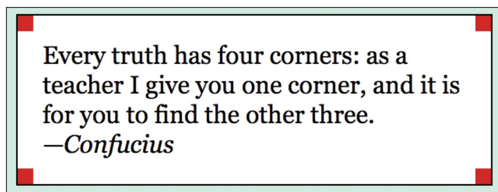


图5-19　把b元素放到四角

正如你所看到的，每一个都在它们各自的位置上，并且生成了一个20px × 20px的小框。这时已经可以看出一个问题了：它们分别位于各自的角落里，而红色的背景应该覆盖div的边框，如图5-20所示。因此：

```
b.tl {top: 0; left: 0; margin: -2px 0 0 -2px;}
b.tr {top: 0; right: 0; margin: -2px -2px 0 0;}
b.bl {bottom: 0; left: 0; margin: 0 0 -2px -2px;}
b.br {bottom: 0; right: 0; margin: 0 -2px -2px 0;}
```

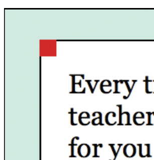


图5-20　新的放置位置（特写）

这会把每个b元素向外拉一些，刚好使它们可以盖住div的边框。当然，如果div有更粗的边框，那么你应该相应地把b元素向外拉出相应的距离。

现在你需要的就是一个可以填充各个角的图像了。我的确指的是图像，而且只有一个图像，大致如图5-21所示。

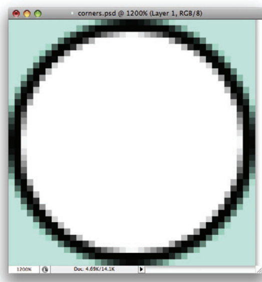


图5-21　用来创建圆角的完整图像

我会帮你省去计算的麻烦：图像是40 px × 40 px大小的，且实际上是个半透明的PNG图像，上面有个打孔的圆圈和边框，且圆圈周围设置成了跟页面总体背景一致的颜色。为了清晰起见，我们将这个图像命名为corners.png。

那么现在可以把CSS改成：

```
b.c {position: absolute; height: 20px; width: 20px;
background: url(corners.png) no-repeat;}
```

这就是这里所需的全部改动了，背景色的值默认是transparent、背景附着的值默认为scroll，而0 0被默认分配给了position。

现在也是删除红色背景规则的好时机，尽管严格地说并不是很必要，因为这条规则里隐藏的transparent值会将它覆盖掉。

现在，改变b元素的样式使之按照我们的需要与图像对齐（如图5-22所示）：

```
b.tl {top: 0; left: 0; margin: -2px 0 0 -2px;
background-position: top left;}
b.tr {top: 0; right: 0; margin: -2px -2px 0 0;
background-position: top right;}
b.bl {bottom: 0; left: 0; margin: 0 0 -2px -2px;
background-position: bottom left;}
b.br {bottom: 0; right: 0; margin: 0 -2px -2px 0;
background-position: bottom right;}
```

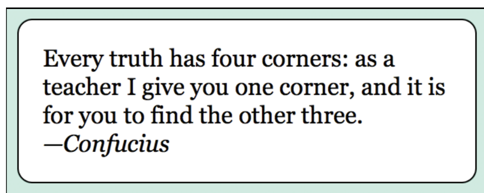


图5-22 填充到四角的图像创建了圆角

就像这样，圆角出来了。

这种技术最棒的地方就是，你不仅可以创建向外弯曲的边框，还可以轻松创建扇形角，或者斜切角，尽管发挥你的想象力吧（图5-23中有些例子）。你只需要更换用来创建这些角的图像，同时也可能需要调整一下b元素的大小。

此外，并没有人强迫你总是使用4个角。如果你只需要让两个角变成圆角，那么只包含两个相关的b元素即可。例如，对于一个页面底部的页脚，你可能只希望让上面两个角变成圆角，那么：

```
<div class="rounded footer">
(…content…)
<b class="c tl"></b>
<b class="c tr"></b>
</div>
```

没问题!

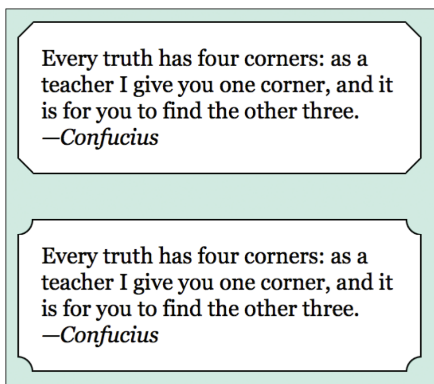


图5-23　一些可选的角

你可能已经注意到了，我把b元素放在了需要圆角化的元素的内容后面。由于它们是使用绝对定位放置的，因此它们在需要圆角化的元素中的位置其实并不重要。它们可以放在最前面、最后面，全部随机混合也可以。因此，把它们放在对你来说最有意义的地方吧。

这种方法有个缺点就是，如果你想改变页面背景色的话，那么圆角的图像也要随之更改，以便与背景色对应。此外，如果整个站点拥有多种不同的背景色，那么你需要为每种可能的颜色准备相应的圆角图像和CSS。还有一个可能更大的缺点就是，如果包围在圆角元素周围的背景不是单一的颜色，比如渐变或者平铺的背景，圆角和周围的背景就无法正确匹配了。使用这种技术时，你只能尽量降低这些情况发生的几率。

还有一个关于历史兼容性的注意事项，即这种方法在IE6下并不能像预期的那样工作，除非你给div指定一个明确的宽度。你可以使用像素、em、百分比或者其他的单位，但是如果坚持使用默认值auto的话，底部的圆角就不会在它们应在的地方了。这只是个小困扰，但却是应该了解的。IE6也不支持PNG的透明，所以你还得准备一个替换用的GIF图像，或者干脆把这些对IE6全都隐藏。反正对于IE6的用户来说，页面上没有圆角也不会使他们太痛苦的。

5.6　CSS3 圆角

一旦你掌握了如何控制曲线的大小，这些事情就真的太简单了。它的优点是纯CSS驱动，不需要额外的标记，而且不需要元素周围的平面颜色（flat-color）。缺点是浏览器的支持有限，而且还需要烦人的供应商前缀，截至撰写本文之时，还没有任何版本的IE浏览器可以支持它们（IE9已经许诺支持了[①]）。

首先，找到5.5节中的图5-22，要想使用CSS3创建同样的基本效果，你只需要这么做：

```
.rounded {background: #FFF; border: 2px solid #000;
border-radius: 20px;}
```

① IE9已经支持CSS3圆角，且不需要供应商前缀，使用border-radius即可。

就这些了！不过（截至撰写本文之时）它在许多浏览器中还都不能正常工作，因为border-radius还没有定稿呢。为了确保之后的几年内它能在支持它的浏览器中正常工作，可以为它们添加供应商前缀，当不需要它们的时候可以直接删掉。

```
.rounded {background: #FFF; border: 2px solid #000;
-moz-border-radius: 20px;
-webkit-border-radius: 20px;
border-radius: 20px;}
```

当IE能够支持圆角（如图5-24所示）的时候，是不是还得声明-ms-border-radius呢？可能吧，这取决于什么时候border-radius能宣布它已经足够稳定以至于可以去掉供应商前缀，以及什么时候IE能添加对它的支持。

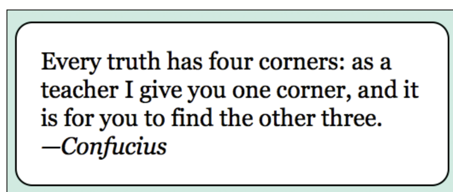


图5-24 非常简单的圆角

这种方法的优点就是，你不需要费力地摆弄前一节那种技术所需的额外的HTML、CSS以及图像，就能使元素的角变成圆角。而页面的背景也可以直接透过圆角，无论是平面颜色、渐变颜色或者全部是格子图案都没问题。

你可以使用两个值改变曲线的形状（如图5-25所示），例如：

```
.rounded { background: #FFF; border: 2px solid #000;
-moz-border-radius: 20px / 60px;
border-radius: 20px / 60px;}
```

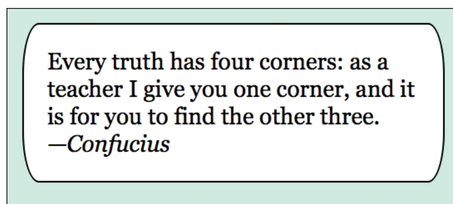


图5-25 椭圆形的角

注意现在这些角并不是完美的圆弧，而是有点儿椭圆的意思了。这就是斜杠分隔的两个值产生的效果。这个斜杠很重要，如果丢掉了斜杠，则会将各个角设置成不同的大小，不过每个都是圆弧。（我去掉了-webkit-这一行，因为在撰写本文之时，WebKit浏览器已经不支持这种写法了。）

假设你故意去掉了斜杠：

```
.rounded { background: #FFF; border: 2px solid #000;
-moz-border-radius: 20px 60px;
```

```
border-radius: 20px 60px;}
```

结果将如图5-26所示。

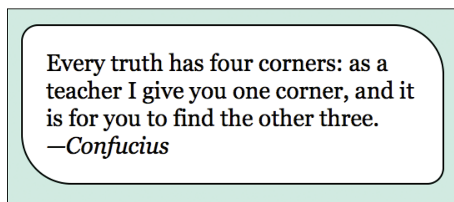


图5-26　半径不等的角

也有一些属性可以用于单独设置每一个角，没有前缀的版本为border-top-right-radius、border-bottom-right-radius、border-bottom-left-radius以及border-top-left-radius。每个属性都可以接受一个或两个值：一个值可以生成圆弧，而两个值可以生成椭圆。斜杠只用在border-radius中，而且很有必要，以便把一个结果（不同大小的圆角）跟另一个（相同大小的椭圆角）区分开来。

事实上，如果想支持WebKit的话，那么每个角的独立属性就很有用了。这是因为尽管它不支持像20px 60px这样的形式，但是它支持每个角的独立属性。因此，为了在基于Gecko和基于WebKit的浏览器中都能得到图5-26中所示的效果，你应该这样写：

```
.rounded { background: #FFF; border: 2px solid #000;
-webkit-border-radius: 20px;
-webkit-border-top-right-radius: 60px;
-webkit-border-bottom-left-radius: 60px;
-moz-border-radius: 20px 60px;
border-radius: 20px 60px;}
```

很丑，但是很有效。

5.7　CSS 精灵

这种技术最初是由Dave Shea（因CSS禅意花园而闻名）在2004年提出并开始流行的，CSS精灵（CSS sprite）是一种可以实现快速悬停效果的技术。现在已经演变成了通过将装饰性的图片合并下载，从而降低服务器负载的技术。

CSS精灵最基本的例子就是包含两种状态的图标，即一个挨着链接的正常显示版本，一个当链接被悬停时的“点亮”版本。图像看上去如图5-27所示。

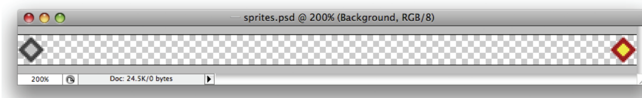


图5-27　精灵图

这里两个图标之间的空白间隔是有原因的，一会儿你就会看到了。通过一点儿CSS，可以让图标在导航条中挨着链接显示。

```
.navbar li a {background: url(sprites.png) 5px 50% no-repeat;
padding-left: 30px;}
```

这可以将它们放在链接的垂直中点上，一直延伸到左边缘。现在，要想使这个图标在链接被悬停时被“点亮”（如图5-28所示），需要改变背景图像的位置。

```
.navbar li a:hover {background-position: -395px 50%;}
```

图5-28　悬停和未悬停时的图标（另见彩插图5-28）

负的水平偏移量的作用是将背景图像向左拉过395 px，这是sprites.png中400 px的空白减去原来规则中5 px的偏移量得到的。由于图标的“点亮”版本距离背景图像的左边缘400 px，因此它刚好落在相同的地方。

这可以扩展为任意多种状态的链接，直至全部。你可以为未访问的、访问的、悬停的、获得焦点的以及激活的链接都设置不同的图标（如图5-29所示）：

```
.navbar li a:link {background-position: 5px 50%;}
.navbar li a:visited {background-position: -395px 50%;}
.navbar li a:hover {background-position: -795px 50%;}
.navbar li a:focus {background-position: -1195px 50%;}
.navbar li a:active {background-position: -1595px 50%;}
```

为此，你可以设置一个包含条纹以及各种不同链接状态图标的图像，只需要把每个图标安置在自己的条纹中并留出足够的空间，防止它们在其他链接中出现。

在这种情况下，你需要分别为每种类型设置垂直的偏移量（以像素为单位），这里有个代码片段可以表明我的意思。

```
.navbar li a.internal:link {background-position: 0 0;}
.navbar li a.external:link {background-position: 50px 0;}
.navbar li a.internal:visited {background-position: 0 -400px;}
.navbar li a.external:visited{background-position: 50px -400px;}
```

图5-29　各种链接状态的精灵图标

你或许以为这会导致这个图像比那些独立的图像大两倍以上，但实际上不是的。由于GIF算法的工作原理，假设所有这些干扰像素都是同样的（缺少）颜色（如本例所示），那么不管这些变体图标的间隔是4 px还是4000 px，文件的大小基本上都是相同的。一旦http头的多余大小以及服务器处理两个连接（每个图像一个连接）时的负载成为影响你网站速度的因子，那么使用精灵将会很高效。

这种洞察力很关键，它可以使你理解为什么有些站点竟然把所有的图标、圆角以及其他装饰性的小图全都塞在一个大图中，然后再简单地使用background-position来显示需要的图标。

这样的事情可能会过度地杀伤你的站点，但回头再看看你的设计，你或许会比第一次看它的时候发现更多可以使用精灵的地方。

5.8　滑动门

这种技术起初是由Doug Bowman（因对Wired站点进行全CSS的重设计而闻名）在2003年提出并开始流行的，“滑动门”（Sliding Doors）是一种可以使文本导航链接变成花哨的选项卡的技术。然而，通常的做法都只适用于效果而不适用于选项卡。

图5-30展示了你希望看到的最终效果。

图5-30　最终效果

是的，确实可以直接通过图片来实现，但是改变选项卡上的文本时就很痛苦了，尤其是选项卡有多种状态的时候。使用如下标记就容易得多了：

```
<ul class="nav">
<li><a href="index.html">Home</a></li>
<li><a href="products.html">Products</a></li>
```

```
<li><a href="buy.html">Buy!</a></li>
<li><a href="contact.html">Contact</a></li>
</ul>
```

那么，如果“Buy!”变成了“Checkout”或者“Store”的话，你就只需要更新标记中的文本了。

好了，这是不错，但是那些选项卡怎么办？好吧，首先你需要一个选项卡的大图。真的，是一个大图，要像图5-31中所示的那么大。

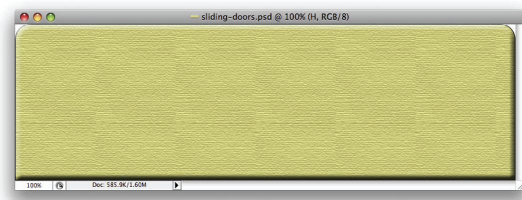


图5-31　选项卡的大图（另见彩插图5-31）

然后把它切成两片——左侧的窄条以及其余的部分，如图5-32所示。

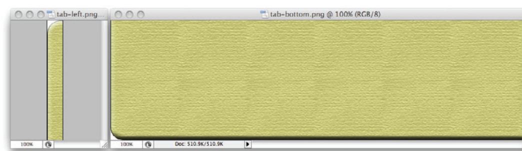


图5-32　每个选项卡的两扇“门”（另见彩插图5-32）

不管你信不信，这就是全部所需的跟图像相关的东西了。现在你需要恰当的CSS（最终效果见图5-33）：

```
ul.nav, ul.nav li {float: left; margin: 0; padding: 0; list-style: none;}
ul.nav {width: 100%;}
ul.nav li {background: url(tab-right.png) no-repeat 100% 0;}
ul.nav li a {background: url(tab-left.png) no-repeat;
display: block; padding: 10px 25px 5px;
font: bold 1em sans-serif; text-decoration: none; color: #000;}
```

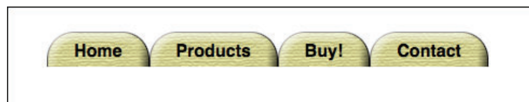


图5-33　最终结果

这就完事了，你已经拥有选项卡了！

如果你临时移除链接背景的话，就很容易看出来它是如何工作的。一旦这些图像被去掉，你就能看到选项卡图像的右半部分填充了整个列表项。当然，链接还在其中，因此当把链接的左边

添上选项卡图像左侧的小条时，它就会位于列表项背景左侧的上方。

现在，假设你希望选项卡在鼠标悬停时被点亮，那么有两种方法可以实现。两种方法都使用了任意元素的悬停，其中最简单的就是交换图像。

```
ul.nav li:hover {background-image: url(tab-right-hover.png);}
ul.nav li:hover a {background-image: url(tab-left-hover.png); color: #FFF;}
```

这里的缺点就是，当页面载入后选项卡第一次被悬停时，从服务器获取图像会稍有延迟。为了避免这个问题，可以结合使用CSS精灵技术（见5.7节）。现在的选项卡看起来如图5-34所示。

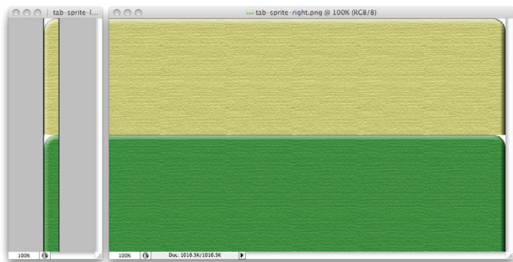


图5-34　选项卡精灵图的两半（另见彩插图5-34）

悬停效果的CSS看起来像这样：

```
ul.nav li:hover {background-position: 100% 400px;}
ul.nav li:hover a {background-image: 0 400px;}
```

将这整个实现思路应用在侧边栏也是很可行的，可以实现沿着设计的侧边放置的可伸缩选项卡。在那种情况下，你需要把选项卡的大图横着切开，而不能竖着切了，如图5-35所示。

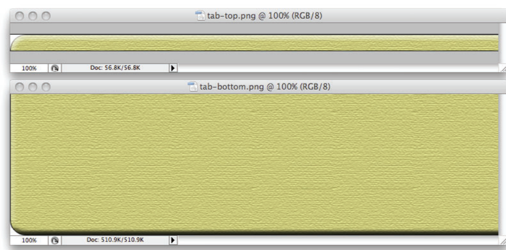


图5-35　水平选项卡组的两半图片（另见彩插图5-35）

然后使用跟之前一样的标记并应用如下样式：

```
ul.nav, ul.nav li {margin: 0; padding: 0; list-style: none;}
ul.nav li {background: url(tab-bottom.png) no-repeat 0 100%;}
ul.nav li a {background: url(tab-top.png) no-repeat 0 0;
display: block; padding: 5px 15px;
font: bold 1em sans-serif; text-decoration: none; color: #000;
display: block;}
```

结果将会如图5-36所示。

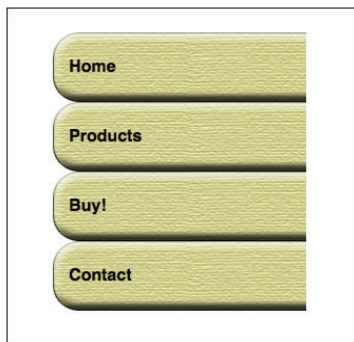


图5-36 横向的效果

对于悬停效果，适用同样的原则：通过CSS精灵来实现即可。

展望未来，终有一天多背景会被广泛支持，那时就可以把所有的选项卡碎片整合到一个元素中了。

5.9 裁切的滑动门

原始的滑动门技术有个缺点，那就是它会强迫你把“页面背景”作为选项卡的一部分。这也没什么，只要选项卡周围是单色的背景而且不会变动就行了。但是，如果想把选项卡放到背景可能发生变化的不同情景中，或者放在图案背景等上的话该怎么办呢？

为了实现这种效果，你需要跟之前一样的一些选项卡碎片，只不过这次要包含透明的部分以使周围的背景可以“透过”它们。图像如图5-37所示，为了简单起见，只使用纯粹的选项卡并去掉悬停效果。

图5-37 选项卡的两半（另见彩插图5-37）

无论是使用GIF89a[①]，还是alpha通道的PNG，都取决于你和你站点的受众。我是使用PNG的，因为它们可以创建平滑的透明边缘。

现在，如果你使用跟之前讲过的技术一样的标记和CSS的话，那么就会得到如图5-38所示的结果。

① GIF89a版，1989年7月发行。

图5-38　把图像放到之前技术所用的标记中所产生的结果

好吧，我们已经成功一半了。每个选项卡的右侧还好，不过我们可以看到左侧列表项元素（li）的背景图像会透过链接元素（a）背景的透明部分，包括左上角！

想搞定这个就需要一点儿技巧了。首先，将链接元素向左拉出列表项元素一些。这可以通过很多种方法实现，或许最简单的方法就是使用相对定位使它们向左偏移（如图5-39所示），并确保有足够的空间能放得下它们。

```
ul.nav, ul.nav li {float: left; margin: 0; padding: 0; list-style: none;}
ul.nav {width: 100%;}
ul.nav li {background: url(tab-clip-right.gif) no-repeat 100% 0;
margin-right: 25px;}
ul.nav li a {background: url(tab-clip-left.gif) no-repeat;
display: block; padding: 10px 0 5px 25px;
position: relative; left: -25px;}
```

图5-39　使裁切的选项卡并肩排列

看到我做了什么吗？每个链接元素都被向左移动了25 px。只做这些还不够，因为这意味着从第二个到最后一个链接都会覆盖它们前面的链接（列表项）。给这些列表项设置25 px的右侧外边距可以为链接元素打开足够的空间，防止它们之间出现重叠或空白。

然而，这里还有个小问题：每个选项卡的右侧将变得不可单击，因为链接已经被往左移动了（仔细看图5-39中，其中最后一个选项卡上的鼠标指针还是默认的）。因此，更好的方法就是使用一点儿外边距的小技巧。在这种情况下，把最后一条规则改成：

```
ul.nav li a {background: url(tab-clip-left.gif) no-repeat;
display: block; padding: 10px 25px 5px; margin-left: -25px;}
```

通过这样的设置，每个超链接左侧的边缘都被向左拉出25 px，即超出列表项左侧边缘25 px。这使得链接能够覆盖之前列表项右侧伸出的外边距部分，就像使用相对定位的方法一样。不过这一次，链接的右侧边缘是紧贴着列表项的右侧边缘的，而不是被移走了，因此选项卡可以按照预期工作。正如图5-40中所展示的，指针已经变成了手形指针，而裁切的角使得页面背景可以穿透过来！

图5-40　运作良好的裁切选项卡

5.10 CSS 视差

CSS视差（CSS parallax）是一种很精巧的技术，我们可以把它当做站点上的一个复活节彩蛋来实现，同时它也能让我们了解基于百分比的背景图像定位可以那么简单而直接地创造出意想不到的效果（这种效果很难通过打印展示，因此你最好亲自尝试一下这种技术）。

首先，考虑一下百分比定位是怎么实现的。假设你把一个背景图像的位置设置为`50% 50%`，那么它的中心将与背景区域的中心对齐。类似地，如果设置为`100% 100%`，那么它的右下角就会和背景区域的右下角对齐。图5-41是两种不同的放置方法示例。

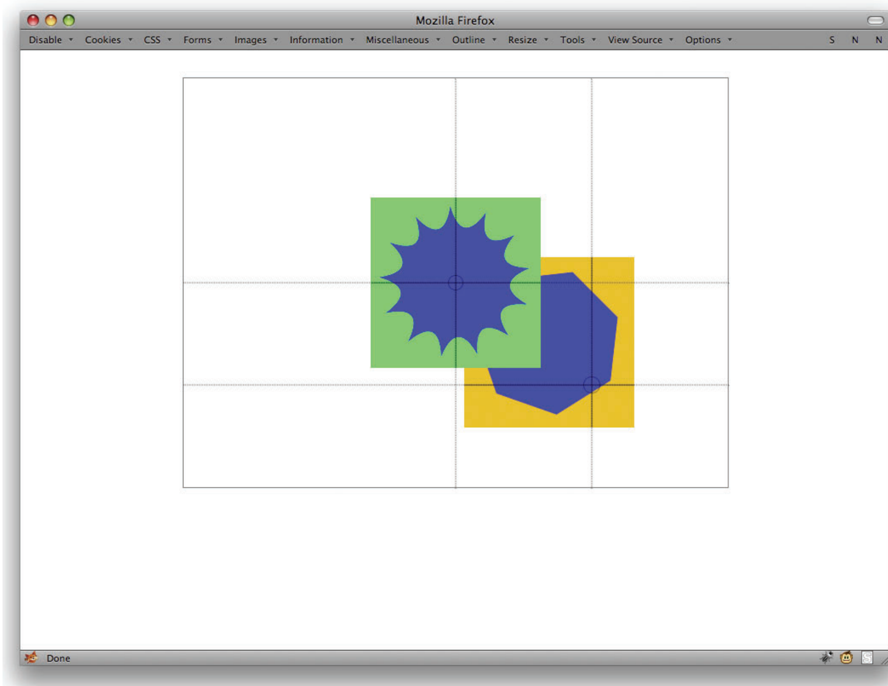


图5-41　设置为`50% 50%`和`75% 75%`时的图示（另见彩插图5-41）

这就意味着背景图像位置的百分比值实际上被使用了两次。第一次用于找到背景区域中所定义的点，第二次用于找到图像本身中定义的点，然后再把这两个点对齐。

那么，当背景区域的大小动态改变时会怎样呢？以这个规则为例：

```
body {background: url(ice-1.png) 75% 0 no-repeat; width: 100%;
padding: 0; margin: 0;}
```

进一步假设ice-1.png为400 px宽，那么在一个恰好为800 px宽的浏览器窗口中，从ice-1.png左边数第300个像素就会跟从`body`的左边数第600个像素对齐，如图5-42所示。

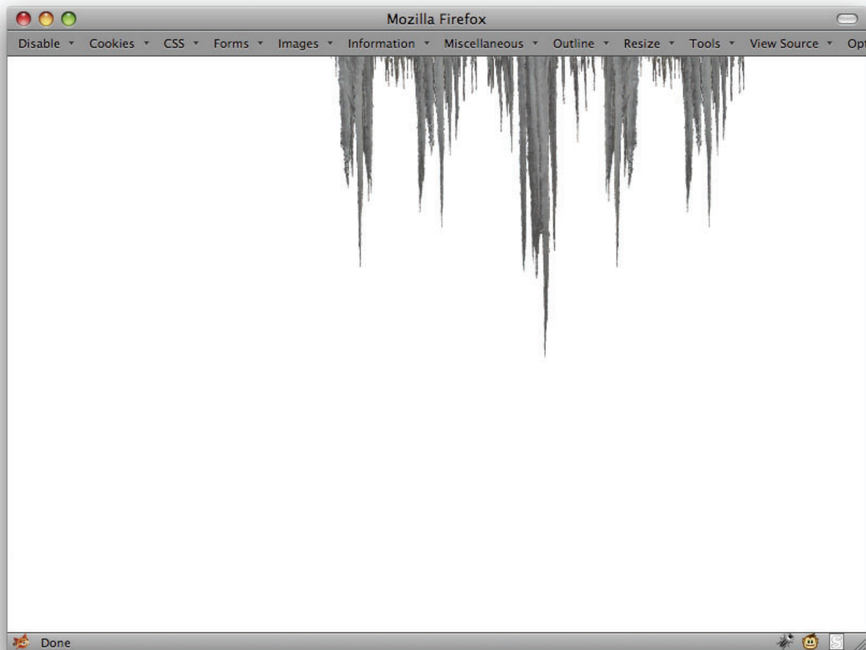


图5-42　冰柱已就位

现在想象一下，当浏览器窗口变窄（因而body也变窄）时冰柱会产生什么效果。冰柱会相对于页面布局向左移动一些，当然，仍然是对应body向左移动之后75%的点。如果窗口和body变宽的话，则冰柱就会向右侧移动了。

现在考虑一下，如果把图像的水平放置位置改成50%的话会发生什么情况。那会使图像在body中居中，而且图像的移动速率会比设置为75%时低。如果把它降到0%，即挨着body的左边缘放置的话，那么在body的大小改变时图像就根本不会移动了（相对于整个页面的布局来说）。

现在假设有两个背景图，其中一个左对齐，而另一个在75%的位置，两个图像都是水平平铺的（如图5-43所示），例如：

```
body {background: url(ice-1.png) 0 0 repeat-x; width: 100%;
padding: 0; margin: 0;}
div#main {background: url(ice-2.png) 75% 0 repeat-x; width: 100%;}
```

很漂亮的分层效果。然而更重要的一点是，当窗口变窄或者变宽时，在div#main上的冰柱会滑过body上的冰柱。这就是潜在的有意思的地方，不过首先让我们更进一步，把body的背景移动一下使它不再是左对齐。

```
body {background: url(ice-1.png) 25% 0 repeat-x; width: 100%;
padding: 0; margin: 0;}
div#main {background: url(ice-2.png) 75% 0 repeat-x; width: 100%;}
```

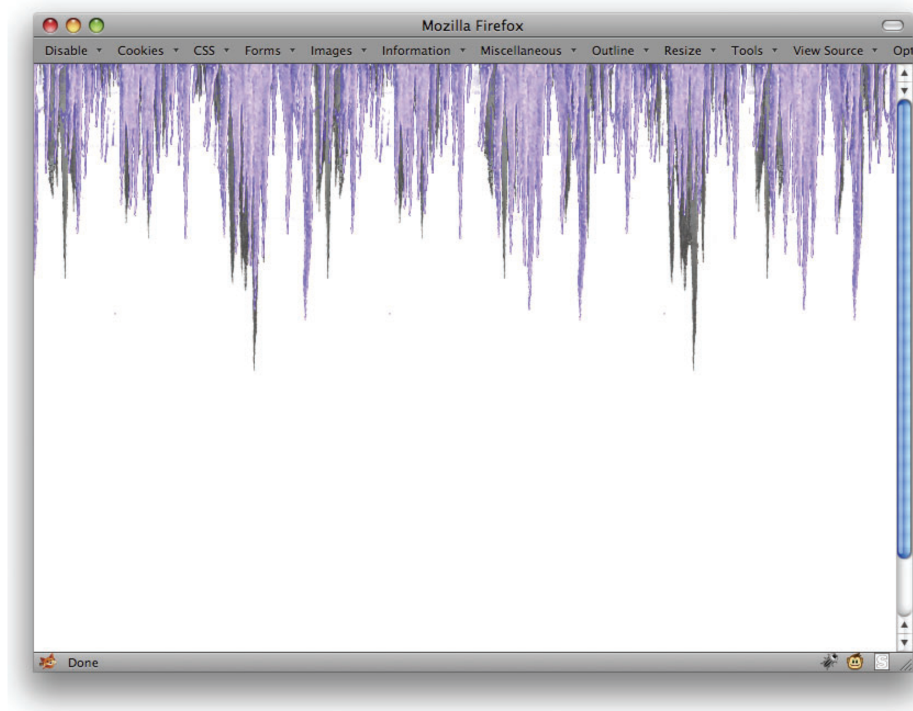


图5-43　两组冰柱（另见彩插图5-43）

现在，当浏览器窗口改变尺寸时两组冰柱都会移动，但是速率却不相同。事实上，body上冰柱的移动速率是其尺寸改变速率的1/4，而div#main上冰柱的移动速率是其尺寸改变速率的3/4。那么，如果以12 px/s的速率改变窗口的尺寸，那么body的背景就会以3 px/s的速率移动，而div#main的背景会以9 px/s的速率移动。

因此，如果希望背景移动的速率大于尺寸改变的速率，那么应该给水平的偏移量设置一个大于100%的百分比值。假设想让背景图像沿着body的顶部移动的话，那么要使背景图像移动的速率为尺寸改变速率的两倍，因而应使用200% 0的background-position（背景定位值）。若想使其沿着body的底部移动，那么这个值应该为200% 100%，而使其在body中垂直居中的话则应该设为200% 50%。

现在，为了能够产生真实的视差感觉，我们可以使用负的百分比值，来让图像朝着与窗口尺寸改变方向相反的方向移动。

因此，与在浏览器窗口变宽时让背景图像向右移动、在浏览器变窄时让图像向左移动不同，你可以实现相反的效果。例如：

```
body {background: url(ice-1.png) -75% 0 repeat-x; width: 100%;
padding: 0; margin: 0;}
div#main {background: url(ice-2.png) 75% 0 repeat-x; width: 100%;}
```

5

通过这样的设置，当浏览器窗口变宽时冰柱看起来就会是朝着远离窗口中心的方向移动的，这会产生类似“放大”的效果。而当窗口变窄时，冰柱会朝向窗口的中心移动，类似“缩小”的效果。

5.11　参差浮动

许多设计师的愿望清单上都有这样一件事，那就是希望能够使文本沿着不规则的形状浮动，而不想每天都对着元素框。好吧，这不仅是可能的，而且简单又可靠，尽管会使用许多标记。

假设你想让文本沿着一条平缓的倾斜曲线流动放置，如图5-44所示。

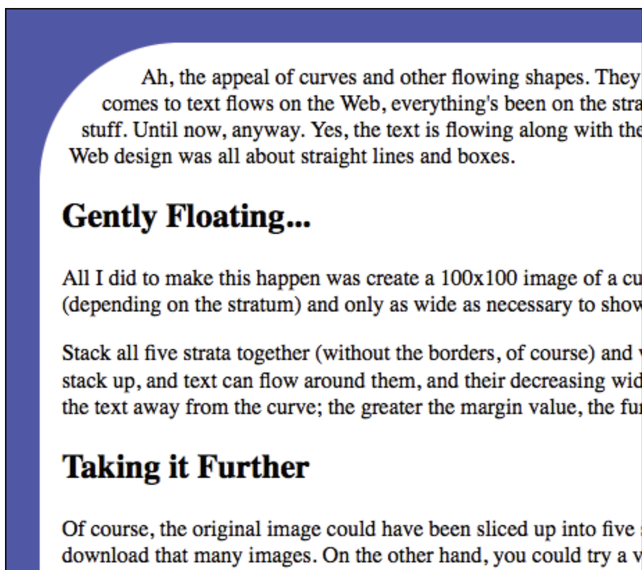


图5-44　文本沿着曲线流动放置的预期效果

乍一看好像不可能，是不是？实际上真的很简单，你需要做的只是把那个曲线切成一堆，然后浮动所有的图像。首先创建20 px高的曲线分片，并且确保它们在曲线外部包含透明区域（如图5-45所示）。

注意每个分片是如何恰好宽到包含曲线的可视部分的，并且不多也不少。现在把这些分片丢到标记中，并放到你希望曲线开始的那个点之前。

```
<div class="curves">
<img src="curve_1.gif" alt="">
<img src="curve_2.gif" alt="">
<img src="curve_3.gif" alt="">
<img src="curve_4.gif" alt="">
<img src="curve_5.gif" alt="">
</div>
```

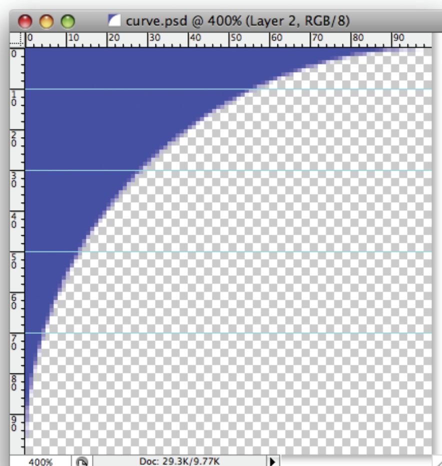


图5-45　带有分片指示的曲线

当然，更多的分片就意味着更多的img元素，相信你已经了解这点了。那么CSS就非常非常简单了：

```
.curves img {float: left; clear: left; margin-right: 1em;}
```

外边距可以防止文本距离分片太近，而且可以按需进行调整。如果为这些图像添加一个临时的边框，如图5-46中左上角所示，你就可以看到浏览器中到底发生了什么。

```
.curves img {border: 1px solid red;} /* temporary */
```

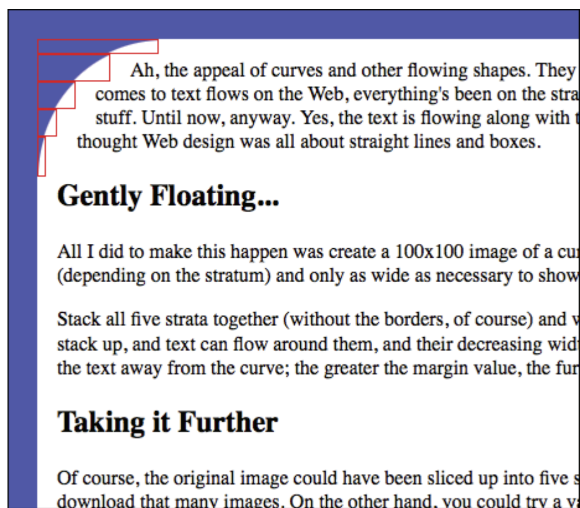


图5-46　开启曲线分片的边框使之可见

这种技术也不局限于简单的曲线。使用同样的方法，在给定任何不规则形状（例如图5-47中所展示的曲线）的情况下，我们都可以使文本流动在它的峰谷之间（如图5-48所示）。

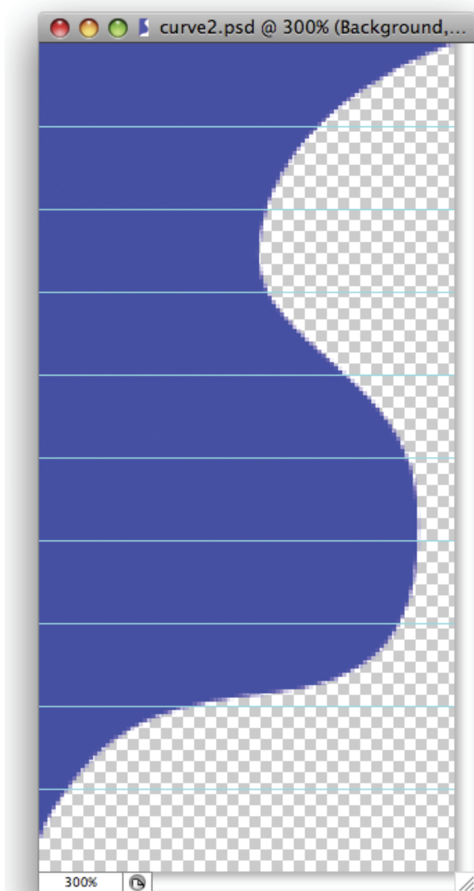


图5-47　带有分片指示的更复杂的曲线

实现该技术的这个版本所使用的CSS与之前的完全相同，只是分片变化了而已。

请注意一件事：从一个分片到下一个分片的宽度变化越激进，在文本与图像之间就越有可能产生重叠。之所以会发生这种情况是因为，浏览器不会检测沿着元素框边缘的每一个像素来确定它们是否覆盖了某个浮动元素。例如，各种浏览器可能只检测框的左上角。如果只是几个像素在一个很宽的浮动图像上，那么那行的文本就会覆盖那个很宽的图像。

当然，创建这些分片有点儿烦人，而且服务器的开销怎么处理呢？幸运的是，这种技术还有一个改进的变体——将在下一节进行阐述。

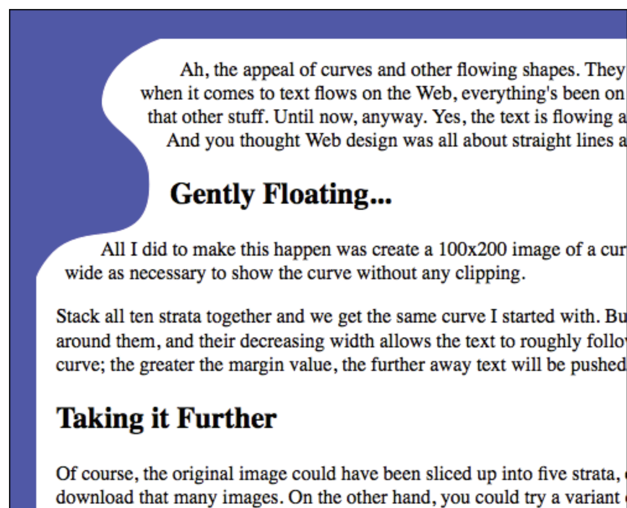


图5-48　浏览器中参差的浮动效果

5.12　更好的参差浮动

在5.11节阐述的“参差浮动”技术的基础上，Nilesh Chaudhari提出了他称为“超级参差浮动”（Super Ragged Floats）的技术，同名文章发表在Evolt上（http://www.evolt.org/article/Super_Ragged_Floats/22/50410/）。Nilesh Chaudhari的见解不是将图像分片，而是把图像放在背景中，然后将透明的框覆盖在上面。这个方法的缺点就是，它要求你在浮动的分片和伴随它们的内容之外包裹div。因此，基于Nilesh Chaudhari的基于原始方法的工作，这里有个可以让你使用较少的自包含标记创建曲线及参差轮廓的技术变体。

首先，考虑前一节中的这些标记，假设你已经把所有的图像都转换成了空的div。

```
<div class="curves">
<div id="sl1"></div>
<div id="sl2"></div>
<div id="sl3"></div>
<div id="sl4"></div>
<div id="sl5"></div>
</div>
```

现在，获取原始的未分片的曲线图像（如图5-49所示）。

此时此刻，你已经有了使文本流呈现曲线所需的全部条件了，只需要在CSS中再添加一些尺寸调整和背景定位就大功告成了（如图5-50所示）：

```
.curves div {float: left; clear: left; margin-right: 20px; height: 20px; width: 100px;
background: url(curve.png) no-repeat;}
.curves #sl2 {width: 42px; background-position: 0 -20px;}
.curves #sl3 {width: 21px; background-position: 0 -40px;}
.curves #sl4 {width: 10px; background-position: 0 -60px;}
```

```
.curves #sl5 {width: 5px; background-position: 0 -80px;}
```

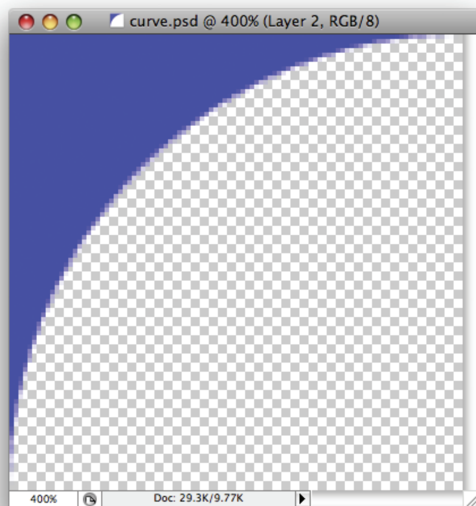


图5-49　Photoshop中展示的曲线图像

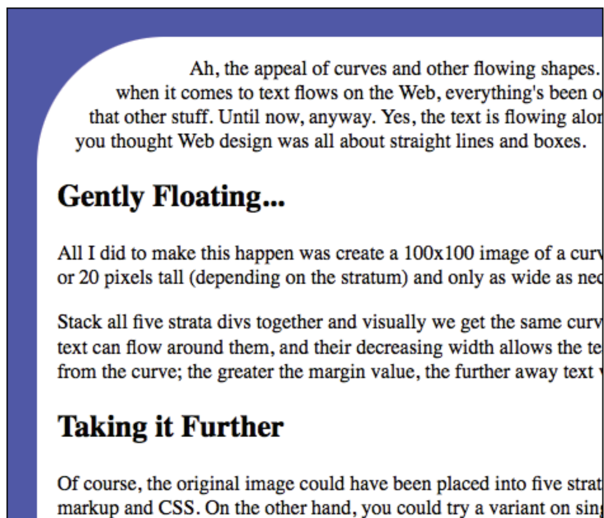


图5-50　对div使用背景定位后在浏览器中呈现出的曲线效果

当然也可以通过嵌入CSS实现这些东西，如果你很想这么做的话。这样的话，所有包含ID的规则以及ID本身就都没有用了，只保留.curves这条规则即可。不过另一方面，将会有一大堆的CSS弄乱你的标记。

```
<div class="curves">
<div></div>
<div style="width: 42px; background-position: 0 -20px;"></div>
<div style="width: 21px; background-position: 0 -40px;"></div>
<div style="width: 10px; background-position: 0 -60px;"></div>
<div style="width: 5px; background-position: 0 -80px;"></div>
</div>
```

选择权在你的手中，请明智选择。

这种技术还有一个技巧（同时也可以应用在上一节展示的分片的版本中）就是，它并不限制所有的div都必须等高。如果曲线有一部分区域是不弯曲的，那么你可以将对应的div伸展到恰当的高度，这会减少所需的元素。你可以使用图像编辑程序事先把流动的框（如图5-51所示）画出来好好计划一下。

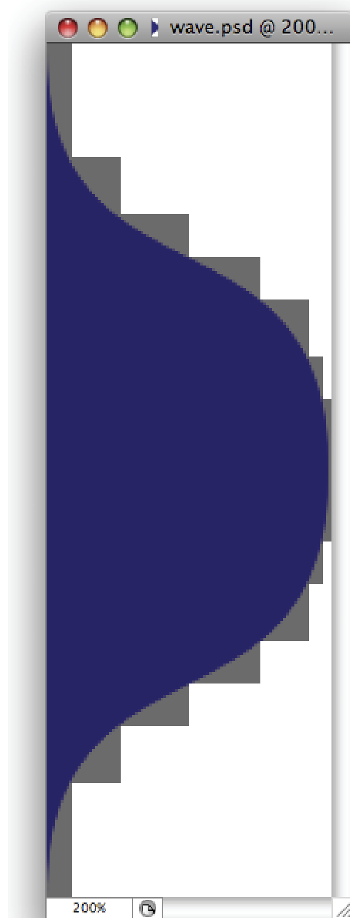


图5-51　Photoshop中展示流动的框

然后框的大小就可以直接复制到你的文档中了。图5-52展示了最终结果。

```
<div class="curves">
<div style="width: 8px; height: 40px;"></div>
<div style="width: 25px; height: 20px; background-position: 0 -40px;"></div>
<div style="width: 50px; height: 15px; background-position: 0 -60px;"></div>
<div style="width: 75px; height: 15px; background-position: 0 -75px;"></div>
<div style="width: 92px; height: 20px; background-position: 0 -90px;"></div>
<div style="width: 97px; height: 15px; background-position: 0 -110px;"></div>
<div style="width: 100px; height: 50px; background-position: 0 -125px;"></div>
<div style="width: 97px; height: 15px; background-position: 0 -175px;"></div>
<div style="width: 92px; height: 20px; background-position: 0 -190px;"></div>
<div style="width: 75px; height: 15px; background-position: 0 -210px;"></div>
<div style="width: 50px; height: 15px; background-position: 0 -225px;"></div>
<div style="width: 25px; height: 20px; background-position: 0 -240px;"></div>
<div style="width: 8px; height: 40px; background-position: 0 -260px;"></div>
</div>
```

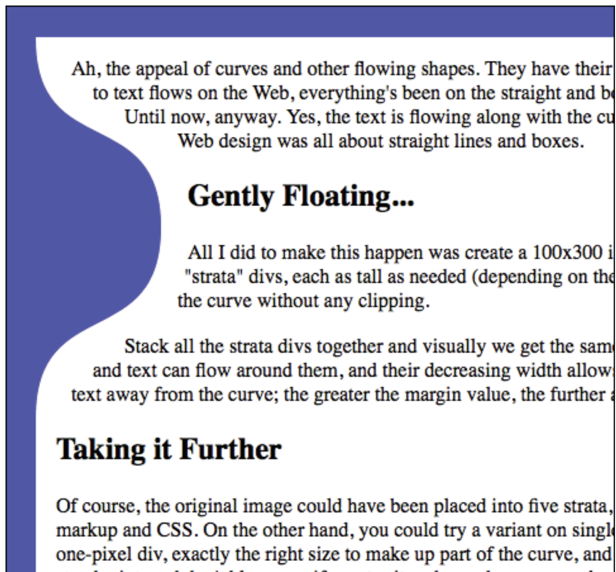


图5-52　通过这个变体技术使波形就位

5.13　图像的框

有些关于图像的东西，大多数人至今都没有意识到：它们与其他元素拥有相同的盒模型（box model），这意味着你可以对图像元素应用诸如背景和内边距等样式。

为什么要纠结这个呢？好吧，以为一个拥有透明部分的方形图标填充背景色的方法为例，它看上去像是这样（如图5-53所示）：

```
img.icon {background-color: #826;}
```

```
img.icon:hover {background-color: #C40;}
```

图5-53　悬停和未悬停状态的图标（另见彩插图5-53）

如图5-54所示，你甚至可以为图像设置背景图像，这可以做出一些有趣的组合效果。

```
img.flake1 {background-image: url(flake1.png) center no-repeat;}
img.flake2 {background-image: url(flake2.png) center no-repeat;}
img.flake3 {background-image: url(flake3.png) center no-repeat;}

<img src="flake-a.png" class="flake1" alt="">
<img src="flake-a.png" class="flake2" alt="">
<img src="flake-a.png" class="flake3" alt="">
<img src="flake-b.png" class="flake1" alt="">
<img src="flake-b.png" class="flake2" alt="">
<img src="flake-b.png" class="flake3" alt="">
```

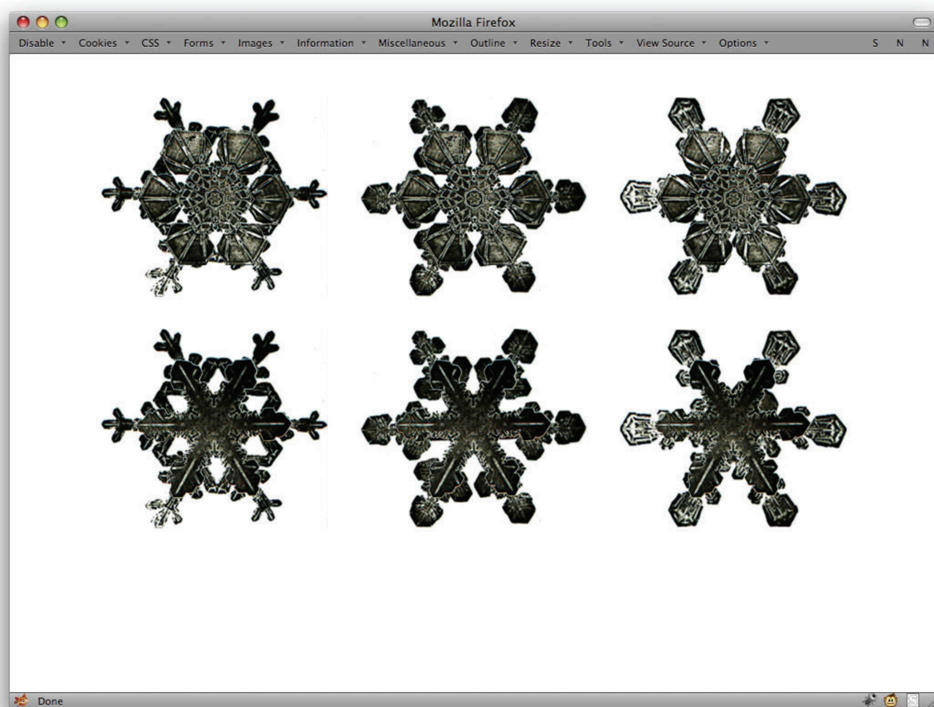


图5-54　组合的雪花

5

内边距也可以很简单地应用在图像上。事实上，通过背景色、边框以及内边距的组合，可以使图像看上去具有深浅不一的双层边框（如图5-55所示）。

```
img.twotone {background: #C40; padding: 5px; border: 5px solid #4C0;}
```

图5-55　通过内边距和边框实现的深浅不一的双层边框（另见彩插图5-55）

再加上轮廓的话，你就会得到类似三层边框的效果了（如图5-56所示）。

```
img.threetone {background: #C40; padding: 5px; border: 5px solid #4C0; outline:
5px solid #40C;}
```

图5-56　通过内边距、边框和轮廓实现的三层深浅不一的边框

5.14　受限的图像

继续上一节的“折腾”好玩图像的主题，这里有个可以使它们在不撑爆其父元素的情况下尽可能变大，或者是强制它们根据原始尺寸按比例放大的方法。这是个非常好用的效果，尤其是当你要在页面中包含照片或者其他可能的大图，又要确保它们不会破坏浏览器中紧凑的布局时。

```
img {max-width: 100%;}
```

这条简单的规则会保证图像不会比包含它们的元素更宽，不过在父元素比这些图像宽的情况下，它们会保持原始尺寸。你可以通过把图像在其父元素中居中来进行强化，像这样：

```
img max-width: 100%; display: block; margin: 0 auto;}
```

图5-57展示了将同一个图像放在3个不同宽度父元素中的例子，其中两个父元素比图像还窄，一个比图像宽（父元素的边缘已通过边框标出）。

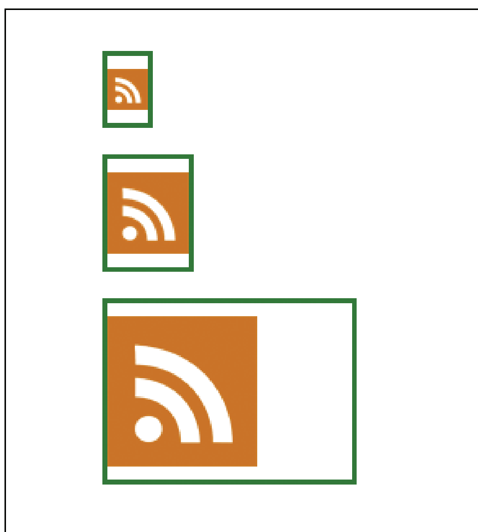


图5-57　相同图像的3个实例

这很明显会使你的图像任由浏览器的缩放操作摆布，因为它们会缩小图像。幸运的是，大多数浏览器已经可以很好地处理这些问题，而不需要太多令人抓狂的、糟糕的人工措施了。

Part 3

第三部分

前沿技术

本部分内容

第 6 章

表　　格

6

我深知这些年你已经无数次地被告知表格是“邪恶的”，任何人都绝对不应该把它们用到页面布局中。很明显这是正确的，表格确实不应该用在布局中。另一方面，对表格进行布局操作其实是一个很精细而又经常被忽略的工作。毕竟，有时候你需要呈现一个数据表，因此对表格过于敷衍了事并不明智。

本章探索一些在样式设计中充分利用表格结构的方法，同时会把表格变成完全不同的样子，譬如地图或柱状图。希望读完本章后，你会发现表格跟其他任何的标记集合都是一样的，都可以给样式设计提供无限的发挥空间。

6.1　表头、主体和脚注

HTML为表格定义了3个元素用于对行进行分组，它们是thead（表头）、tbody（表格主体）和tfoot（脚注）。毫不奇怪，这些元素分别代表了表格的表头、主体部分以及脚注部分。

下面是一个精简的表格结构，使用了其中的两个分组元素。

```
<thead>
<tr>…</tr>
</thead>
<tbody>
<tr>…</tr>
<tr>…</tr>
<tr>…</tr>
<tr>…</tr>
</tbody>
</table>
```

这些元素赋予了表格更多的结构，同时在语义方面也很友好，不过更棒的是你可以使用它们单独对表头里的元素应用与主体不同的样式（如图6-1所示）。因此，你可以使每一列的标题（在thead中）居中对齐而使每一行的标题（在tbody中）右对齐。

```
thead th {text-align: center;}
tbody th {text-align: right;}
```

类似地，你也可以改变各自分组中单元格的颜色、背景、内边距以及其他任何可以对表格单元格应用的样式，恰当地引用那个元素即可。

	Q1	Q2	Q3	Q4
#207	$11,940	$12,348	$14,301	$17,208
#B315	$9,345	$9,834	$10,035	$9,672
#207-B36	$2,787	$3,123	$4,137	$3,711

图6-1　居右和居中对齐不同类型的标题单元格

关于这些行分组元素有个令人惊奇的事实，那就是即使你没有明确地把它们写出来，大部分浏览器也会在DOM（Document Object Model，文档对象模型）中创建它们（如图6-2所示）。在这些浏览器中，下面的规则总是会失效：

```
table > tr {font-weight: bold;}
```

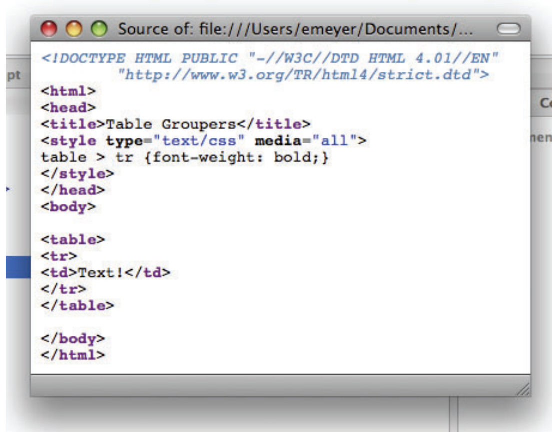


图6-2　如果没有明确写出，浏览器会自动创建一些元素

这是因为在`table`和`tr`之间总是有个`tbody`存在，尤其是当源代码中并没有写分组元素时，浏览器也会自动创建`tbody`。因此你可以把之前那条规则的选择器改写成`table > tbody > tr`，这样就可以匹配任何没有行分组元素的行了。

更加令人惊奇的是，在HTML 4中文档结构中的`tfoot`必须出现在`tbody`之前。HTML 5去掉了这个限制，允许`tfoot`跟随在`tbody`之后，而且各个浏览器也不再强制执行HTML 4的规则了。所以尽管它令人惊奇，但是也算不上难以承受了。

理论上讲，在多视口（multiple viewport）中显示的表头和脚注行应该放到每一个表格片段的顶部和底部。以花哨的规范方式来讲就是，如果你打印一个有好几页长的表格，那么表头和脚注应该放到每一页或者每一页上出现的表格片段的顶部和底部。然而，请注意我使用的词是“理论上讲”。在实际应用中，这是不可能发生的，也许有一天能够做到吧。正如服务员格罗弗[①]所说：

① 《芝麻街》（*Sesame Street*）中的人物。《芝麻街》于1969年首播，是迄今为止获得艾美奖项最多的儿童节目，网址为http://www.sesamestreet.org/。

活着就是希望。

记住，包含thead和tfoot的最大优点之一就是，可以用于对其包含的表格单元格单独应用样式。作为例子，考虑下面的HTML 5片段：

```
<table>
<thead>
<tr>
    <th scope="col">Q1</th>
    <th scope="col">Q2</th>
    <th scope="col">Q3</th>
    <th scope="col">Q4</th>
</tr>
</thead>
<tbody>
    ...
</tbody>
<tfoot>
<tr>
    <td>$83,340</td>
    <td>$87,195</td>
    <td>$91,022</td>
    <td>$90,489</td>
</tr>
</tfoot>
</table>
```

现在，假设你想在列标题的下面划一条线，并且在脚注的汇总数据上面也划一条线，如图6-3所示。那么不需要使用任何的类，只使用表格本身的结构即可。

```
thead th {border-bottom: 1px solid #333; text-align: center; font-weight: bold;}
tfoot th, tfoot td {border-top: 2px solid #666; color: #363;}
```

	Q1	Q2	Q3	Q4
#207-B34	$11,940	$12,348	$14,301	$17,208
#207-B35	$9,345	$9,834	$10,035	$9,672
#207-B36	$2,787	$3,123	$4,137	$3,711
#208-A07	$1,657	$3,003	$2,882	$2,690
#208-A11	$8,947	$7,249	$8,102	$7,821
#208-A12	$9,034	$11,027	$11,793	$10,283
#208-A13	$10,633	$12,574	$12,834	$11,568
#208-A23	$15,856	$16,239	$16,057	$15,712
#209-C17	$8,245	$6,929	$6,498	$5,016
#209-C55	$4,896	$4,869	$4,383	$6,808
Total	$83,340	$87,195	$91,022	$90,489

图6-3 通过边框使表头以及脚注与表格主体区分开来

6.2　行标题

6.1节中提到了行标题。“行标题？”你或许会自言自语，“我以为只有列才会有作为标题的单元格呢。”事实上不是这样的！有一个HTML属性在当初设计的时候就是用来让你指定一个th元素是列标题还是行标题的。

考虑下面的标记：

```
<table>
<thead>
<tr>
<th></th>
<th>Pageviews</th>
<th>Visitors </th>
</tr>
</thead>
<tbody>
<tr>
<th>January 2010</th>
<td>1,367,234</td>
<td>326,578</td>
</tr>
<tr>
<th>February 2010</th>
<td>1,491,262</td>
<td>349,091</td>
</tr>
</tbody>
</table>
```

注意表格主体中的每一行都以th元素开头，那些就是行标题。相信大家都可以推断出它们跟行中后面的数据是相关的。即使浏览器可能也会推断出同样的结论。不过，最好还是像这样明确指出：

```
<table>
<thead>
<tr>
<th></th>
<th scope="col">Pageviews</th>
<th scope="col">Visitors </th>
</tr>
</thead>
<tbody>
<tr>
<th scope="row">January 2010</th>
<td>1,367,234</td>
<td>326,578</td>
</tr>
<tr>
<th scope="row">February 2010</th>
<td>1,491,262</td>
<td>349,091</td>
```

```
</tr>
</tbody>
</table>
```

为th元素恰当地添加已赋值的scope（作用域）属性，相当于明确地告诉浏览器th元素与它周围单元格的关系（如图6-4所示）。

Month	Pageviews	Visitors
January 2010	1,367,234	326,578
February 2010	1,491,262	349,091

Web traffic

图6-4　基于标题单元格的作用域为其应用样式

在可视化的浏览器中，这算不了什么，不过你可以使用属性选择器对这两种类型单独应用样式。

```
th[scope="col"] {border-bottom: 1px solid gray;}
th[scope="row"] {border-right: 1px solid gray;}
```

这可以成为挂载样式时很方便的一个钩子（hook）。在这种特定的情况下，同样的效果也可以通过thead th以及tbody th达到，不过有时候可能会有行标题在thead中或者列标题在tbody中的情况。因此，最好做好两手准备。

在具有朗读功能的浏览器中，理论上通过scope的值使表格的列和行标题与每个单元格的内容相关联，可以使表格更容易理解。因此，当具备朗读功能的浏览器移到之前展示的表格上面时，对于第一行它可能会说"页面浏览量二零一零年一月一百三十六万七千二百三十四——访问量二零一零年一月三十二万六千五百七十八。"[①]（我放破折号的地方可能会被读成"数据单元格"或者类似的东西，不过你明白我的意思。）

再一次地，注意我的用词"理论上"。在这种情况下，至少有些具备朗读功能的浏览器支持使用scope帮助其做决定，不过一般默认是不开启这个功能的。

6.3　面向列的样式

你可能已经习惯了操作表格的行，但是有时候可能需要对表格的列设置样式。相对于使用一些丑陋而复杂的方法，若想简单地对列应用样式就很难了。（你明白我的意思吗？）

使用标记实现的最简单方法就是应用col元素，以下面这个简单的表格为例。

```
<table>
<col span="2" />
<col />
<col />
<tbody>
<tr>
```

① 浏览器朗读的是英文，这里是译文，目前还没有如此智能的中文屏幕阅读器。

```
        <td>Row 1 cell 1</td>
        <td>Row 1 cell 2</td>
        <td>Row 1 cell 3</td>
        <td>Row 1 cell 4</td>
        <td>Row 1 cell 5</td>
    </tr>
    <tr>
        <td>Row 2 cell 1</td>
        <td>Row 2 cell 2</td>
        <td>Row 2 cell 3</td>
        <td>Row 2 cell 4</td>
        <td>Row 1 cell 5</td>
    </tr>
    </tbody>
    </table>
```

这会设置3个列，其中一个“跨越”了每行的两个单元格，另外两个包含每行的一个单元格。这些加起来是每行4个单元格，而你可以看到现在每行为5个单元格。这意味着每行的最后一个单元格不属于列结构。

那么如何为已有的各列设置样式呢（如图6-5所示）？这似乎非常简单，只需要给col元素应用CSS。

```
col {background: red; width: 10em;}
```

Row 1 cell 1	Row 1 cell 2	Row 1 cell 3	Row 1 cell 4	Row 1 cell 5
Row 2 cell 1	Row 2 cell 2	Row 2 cell 3	Row 2 cell 4	Row 1 cell 5

图6-5　对列元素应用样式

这个非常有限的例子几乎在现今可用的任何浏览器中都能良好地工作。如果只是为列设置简单的背景色以及宽度，那么你将畅通无阻。

然而，如果你想对列应用任何其他样式的话，就没那么走运了。这是因为CSS规范只允许除此之外的两个属性应用在表格的列上，即border（边框）和visibility（可见性），而且两者都没能被广泛支持。

在前一种情况中，如果声明了边框，那么各个浏览器并不会以同样的方式绘制它。有些浏览器会在整个列的外面绘制边框，而另外一些则会将边框应用在列本身以及该列所包含的全部单元格上。如果只是设置1 px的实线边框可能还会得到相对不错的结果，但如果更粗或者不是实线的话就完了。它还要求你对所有的表格声明table {border-collapse: collapse;}，如果想让表格表现更加一致的话，这才是更值得了解的。

在后一种情况中，你能做的只是通过设置visibility: collapse隐藏整个列。这很好，不过它并非在所有浏览器中都工作良好，尤其是在Safari和Chrome以及它们的移动版上。

你们当中的有些人可能确信曾经听说过关于在列元素上应用其他CSS属性的事情。因为IE浏览器允许对col元素应用任何CSS属性，所以这完全有可能。其他的浏览器（以及CSS规范本身）

不允许这么做的原因则是一个很长、很曲折，而且坦白地说很烦人的故事了。在这一点上IE确实做了一件大家盼望的好事。

虽然col元素理论上是为列设置样式的简便方法，但是在真实世界中的应用却非常有限。如果你想为列设置样式，就必须更有创造性。通常的办法就是对整个表格的所有单元格应用类。

```
<table>
<tbody>
<tr>
    <td class="c1">Row 1 cell 1</td>
    <td class="c2">Row 1 cell 2</td>
    <td class="c3">Row 1 cell 3</td>
    <td class="c4">Row 1 cell 4</td>
    <td class="c5">Row 1 cell 5</td>
</tr>
<tr>
    <td class="c1">Row 2 cell 1</td>
    <td class="c2">Row 2 cell 2</td>
    <td class="c3">Row 2 cell 3</td>
    <td class="c4">Row 2 cell 4</td>
    <td class="c5">Row 1 cell 5</td>
</tr>
</tbody>
</table>
```

现在，如果你想让某个特定的列拥有红色背景，那么只需要简单地为那些单元格的类书写相应的CSS。这会重建图6-5中所示的效果。

```
.c1, .c2, .c3, .c4 {background: red; width: 10em;}
```

是的，标记和样式都很笨拙。这里的优点就是你可以对表格单元格应用任何其他可以应用的CSS属性（只是外边距除外）。那么如果你想使那些单元格的文本居中并变成斜体的话（如图6-6所示），那就非常简单了。

```
.c1, .c2, .c3, .c4 {background: red; width: 10em;
text-align: center; font-style: italic;}
```

Row 1 cell 1	*Row 1 cell 2*	*Row 1 cell 3*	*Row 1 cell 4*	Row 1 cell 5
Row 2 cell 1	*Row 2 cell 2*	*Row 2 cell 3*	*Row 2 cell 4*	Row 1 cell 5

图6-6　使用类对“列”应用样式

记住，当对col元素应用样式时，这种改变样式的简单方法在非IE浏览器中是行不通的，因为它已经被CSS规范禁止了。

如果你想通过使用标记的方式为某个特定的列添加一圈边框的话（如图6-7所示），那么还需要一点儿工作。你只需要填充这一列中所有单元格的侧边框，然后为这一列顶部和底部单元格的顶部和底部分别设置边框。

```
td.c2 {border: 2px solid #000; border-width: 0 2px;}
tr:first-child td.c2 {border-top-width: 2px;}
```

```
tr:last-child td.c2 {border-bottom-width: 2px;}
```

Row 1 cell 1	Row 1 cell 2	Row 1 cell 3	Row 1 cell 4	Row 1 cell 5
Row 2 cell 1	Row 2 cell 2	Row 2 cell 3	Row 2 cell 4	Row 1 cell 5

图6-7　通过一些组合的方法为“列”设置边框

如果由于使用这些选择器需要处理一些向后兼容问题这点让你有些不舒服，那么就可以多使用类的小伎俩了——适当地为表格的第一行以及最后一行设置类即可。

```
<table>
<tbody>
<tr class="first">
…
</tr>
<tr class="last">
…
</tr>
</tbody>
</table>
```

完成这些之后，你只需要对CSS做一点儿小改动使之可以使用新的钩子。

```
td.c2 {border: 2px solid #000; border-width: 0 2px;}
tr.first td.c2 {border-top-width: 2px;}
tr.last td.c2 {border-bottom-width: 2px;}
```

让我们再往回退一点儿，考虑一下某种非常规的对列应用样式的方法，即一种完全不需要使用类的方法。首先，把标记中的这些类去除。

```
<table>
<tbody>
<tr>
    <td>Row 1 cell 1</td>
…
    <td>Row 1 cell 5</td>
</tr>
<tr>
    <td>Row 2 cell 1</td>
…
    <td>Row 1 cell 5</td>
</tr>
</tbody>
</table>
```

现在，你将如何对第二列应用样式呢？通过`:first-child`和相邻兄弟选择器就可以实现。

```
td:first-child + td {border: 2px solid #000; border-width: 0 2px;}
tr:first-child td:first-child + td {border-top-width: 2px;}
tr:last-child td:first-child + td {border-bottom-width: 2px;}
```

通过这种方式，意味着为第一列设置样式的话只使用`td:first-child`即可（因为你已经选择了表格中所有作为`tr`第一个子元素的单元格），任何跟随其后的列都可以通过添加n–1个`+ td`

进行选择。那么如果你想将边框移到第四列的话（如图6-8所示），需要建立如下规则：

```
td:first-child + td +td + td {border: 2px solid #000; border-width: 0 2px;}
tr:first-child td:first-child + td + td + td {border-top-width: 2px;}
tr:last-child td:first-child + td + td + td {border-bottom-width: 2px;}
```

Row 1 cell 1	Row 1 cell 2	Row 1 cell 3	Row 1 cell 4	Row 1 cell 5
Row 2 cell 1	Row 2 cell 2	Row 2 cell 3	Row 2 cell 4	Row 1 cell 5

图6-8　使用子选择器和兄弟选择器为“列”应用样式

笨拙还是优雅？我想，这取决于你的审美观了。

使用这种方法时要注意的一点是，它会被任何跨越多列的表格单元格破坏。不过那样的话，如果你已经让单元格跨越多列了，那么可能根本不会应用整列的样式了（好吧，你是这样的，但大多数人可能不是）。如果需要处理跨列的单元格，那么使用类的方式可能是你的最佳选择。

6.4　表的映射

许多时候你可能会有一大堆具有地理相关性的数据，比如各州的销售数据、分地区的投票数据等。这些数据通常最好以表格的形式进行组织，但这并不意味着必须用同样的方式来展现它们。为什么不把它放到一个地图中呢？

为了实现这个思路，除了CSS之外还需要两样东西。首先需要给标记添加恰当的类和ID，下面是一个美国各州的例子及其相关标记。

```
<table>
<thead>
<tr>
<th scope="col">State</th>
<th scope="col">Representatives</th>
</tr>
</thead>
<tbody>
<tr id="AL">
<th scope="row">AL</th>
<td>7</td>
</tr>
<tr id="AK">
<th scope="row">AK</th>
<td>1</td>
</tr>
…
<tr id="WY">
<th scope="row">WY</th>
<td>1</td>
</tr>
</tbody>
</table>
```

那么现在还需要连同另外47行[1]数据放在地图上，因而你需要的第二样东西就是：一个地图的图像（如图6-9所示）。

图6-9　地图

有了这个地图，你就可以开始考虑该把数据放在哪里了。在这种情况下，你需要的就是获取每个州的近似中点。我的做法就是在类似Photoshop这样的图像编辑器中打开图像，然后用它来计算出每个点的*X*、*Y*坐标，再把这些坐标写到一个列表中。因此会得到：

AL	692	448
AK	210	560
…		
WY	330	300

这些就是你要放置每个数据的点，不过不要仅停留在简单的像素值上，这些值需要转换为相对于图像像素尺寸的百分比大小。这个地图是1000 px × 700 px的，因此每个水平度量应该除以1000，而每个垂直度量应该除以700，那么就会得到：

AL	69.2%	64%
AK	21%	80%
…		
WY	33%	42.9%

① 美国共有50个州，标记中写出了3个，故还需要47行。

我们先把这个列表放在一边，因为现在要开始书写CSS使它运作起来。首先，确保地图可以正确展现：

```
table, table * {margin: 0; padding: 0; font: 1em/1 sans-serif;}
table {display: block; width: 1000px; height: 700px;
    background: url(us-state-map.gif) no-repeat;}
```

好了，地图已经就位了，不过数据全部都在左侧，并没有伸展开（如图6-10所示）。有意思吧？这是因为所有的表格单元格仍然都以表格的方式悬挂在一起，而table元素本身的行为已经不再像通常的表格行为了。它现在生成了一个块级框，就像div那样。因此，table元素与表格各行以及单元格等之间的布局关联已经被打破了。是的，确实是这样的。

图6-10　地图及数据

你需要把所有的数据都放到指定的位置上。第一步就是获取全部数据来生成块级框并且对它们全部进行定位（理论上，给它们定位就会使它们生成块级框，不过我喜欢明确地声明对display属性的改变，只是为了保险）。你还应该添加一个临时的边框，这样才能看到事情进展如何（如图6-11所示）。

```
tr, th, td {display: block;}
tr {position: absolute; top: 0; left: 0;
    color: #527435;
border: 1px dotted red;}
```

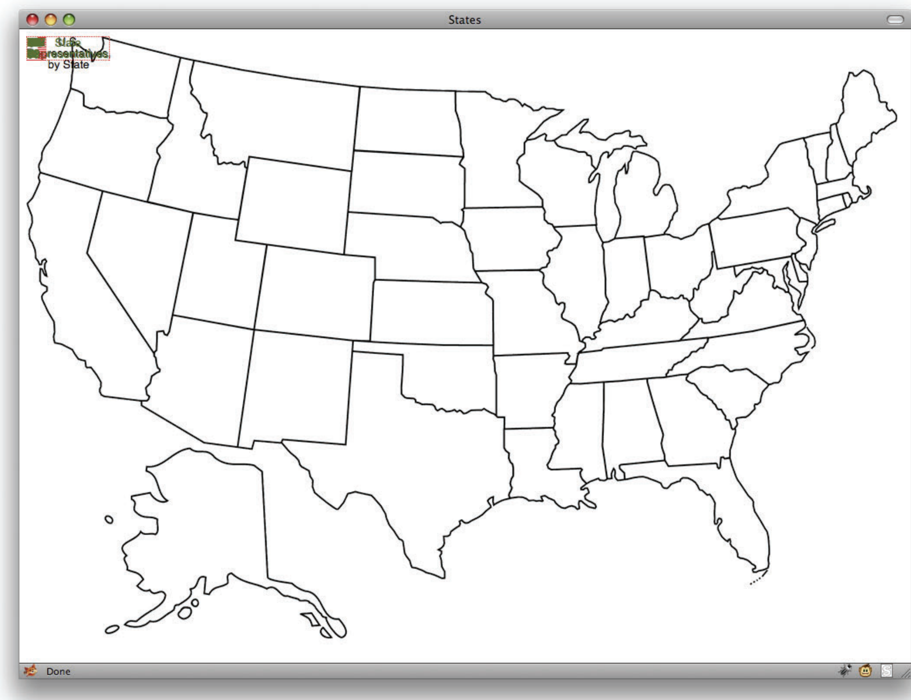


图6-11　第一步，给数据定位

呀，它们全都堆在了地图的左上角，并没有在各自恰当的点上。这时就用到了刚才那个百分比的列表了。每个水平的百分比都会变成left的值，而每个垂直的百分比都会变成top的值（如图6-12所示）。

```
#AL {left: 69.2%; top: 64%;}
#AK {left: 21%; top: 80%;}
…
#WY {left: 33%; top: 42.5%;}
```

好吧，应该说是大部分就位了。它们没有排列整齐的原因是，每个州的中点（或者至少是一些合理的点）都是在你写下列表时选择的。这些值作为顶部和左侧的偏移量，用来将每个定位元素的左上角放置在那些选择的点上。因此定位的tr元素会位于那些点的底部的右侧。

对于所有那些较小的东北部的各州，它们的数据相互重叠了，不过那是待会儿要处理的事件。

解决这个问题的最简单办法就是为每个tr指定宽和高，然后把它们向上和向左拉过刚才指定尺寸的一半距离。下面是一个小实验（如图6-13所示）：

```
tr {position: absolute; top: 0; left: 0;
    width: 2em; height: 2em;
    margin-left: -1em; margin-top: -1em;
    color: #527435;
    border: 1px dotted red;}
```

6

图6-12　把数据放到每个州对应的位置中

图6-13　调整每个数据框的放置位置

嘿，看看这些吧！这些数据已经差不多都在你想要的位置上了。这里当然还有一些可调整的余地，例如佛罗里达的框貌似往下了一点，不过它们已经很好地相对摆放在一起了，不过东北地区除外。文本似乎还有些对齐问题，那么让我们来清理一下这些问题以及其他的一些问题。

```
tr {position: absolute; top: 0; left: 0;
    width: 2em; height: 2.2em;
    margin-left: -1em; margin-top: -1.1em;
    color: #527435;}
tbody th, tbody td {text-align: center;}
tbody th {font-weight: bold; border-bottom: 1px solid gray; margin-bottom: 0.1em;}
```

图6-14比刚才更好了！当然，东北地区的布局仍然过于混乱。

图6-14　清理数据

实际上我打算让它保持这个样子，它可以作为了解为何把数据映射到表格时要格外小心的完美例子。是的，有很多种通过CSS缓解这种问题的方法，比如在那些较小的州中隐藏部分数据框，当鼠标悬停时再显示出来。一个更好的主意就是选取一个每个州都有足够空间放置数据的地图（例如，一个含有东北地区的放大特写、或者有标注线指向较小州的地图，然后数据就可以放在每个标注线的末端了），我把这个想象空间留给你。

还有另外一件需要做的事，它比现有的使用scope属性的方式更能提高表格的可访问性。另一个好处在于，还有其他一些需要放置和应用样式的元素。这个表格还需要一个摘要（summary）和一个表格标题（caption）。

```
<table summary="A list of American states and the number of representatives
```

```
allocated to each in the United States House of Representatives, which is
the lower chamber of the United States Congress.">
    <caption>U.S. Representatives, by State</caption>
<thead>
```

现在你可以使用caption元素并将其作为地图的标题了（如图6-15所示），同时还需要关掉thead的显示，它现在正挂在地图的左上角，看上去很丑。

```
thead {display: none;}
caption {position: absolute;
    top: 0; left: 0; right: 0; text-align: center;
    font: bold 200% sans-serif;}
```

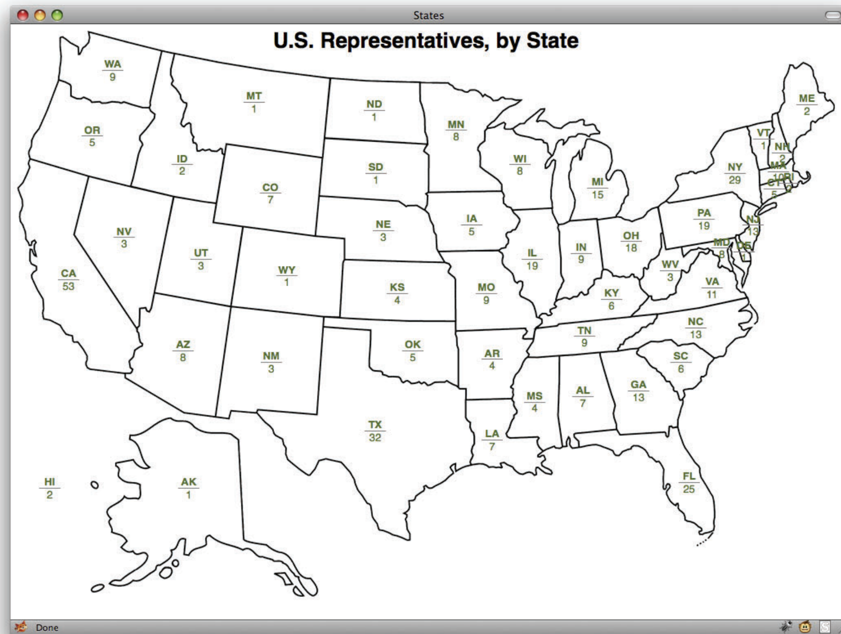


图6-15　使用表格标题并将其作为图表的标题

好极了！就这样，表格的数据都放在了地图上。数据有适当的结构、高度的可访问性以及旧的表格无法提供的视觉享受。

6.5　表的图形化

有时候，数据表本身适合于图形化。它可以是季度利润、过去一周或高或低的温度，或者一年的平均降雨量等，大量的数据集都可以被图形化。

考虑一个我们比较熟悉的情景，这里是一组描述以10天为周期的网站点击量和页面浏览量的数据。

```
<table summary="Server hits and pageviews for meyerweb.com over the period 1/10/10 to
 1/19/10.">
    <caption>Web traffic</caption>
    <thead>
        <tr>
            <th scope="col">Day</th>
            <th scope="col" class="hits">Hits</th>
            <th scope="col" class="views">Views</th>
        </tr>
    </thead>
    <tbody>
        <tr id="day01">
            <th scope="row">1/10/10</th>
            <td class="hits">151,308</td> <td class="views">70,342</td>
        </tr>
        <tr id="day02">
            <th scope="row">1/10/11</th>
            <td class="hits">138,887</td> <td class="views">70,410</td>
        </tr>
        <tr id="day03">
            <th scope="row">1/10/12</th>
            <td class="hits">106,563</td> <td class="views">58,383</td>
        </tr>
        <tr id="day04">
            <th scope="row">1/10/13</th>
            <td class="hits">117,551</td> <td class="views">64,181</td>
        </tr>
        <tr id="day05">
            <th scope="row">1/10/14</th>
            <td class="hits">251,969</td> <td class="views">171,790</td>
        </tr>
        <tr id="day06">
            <th scope="row">1/10/15</th>
            <td class="hits">213,228</td> <td class="views">134,238</td>
        </tr>
        <tr id="day07">
            <th scope="row">1/10/16</th>
            <td class="hits">186,099</td> <td class="views">113,014</td>
        </tr>
        <tr id="day08">
            <th scope="row">1/10/17</th>
            <td class="hits">246,637</td> <td class="views">161,287</td>
        </tr>
        <tr id="day09">
            <th scope="row">1/10/18</th>
            <td class="hits">210,124</td> <td class="views">135,479</td>
        </tr>
        <tr id="day10">
            <th scope="row">1/10/19</th>
            <td class="hits">168,413</td> <td class="views">115,541</td>
        </tr>
    </tbody>
</table>
```

6

在浏览器中渲染之后，原始的数据看起来如图6-16所示。

Web traffic		
Date	**Hits**	**Views**
1/10/10	151,308	70,342
1/11/10	138,887	70,410
1/12/10	106,563	58,383
1/13/10	117,551	64,181
1/14/10	251,969	171,790
1/15/10	213,228	134,238
1/16/10	186,099	113,014
1/17/10	246,637	161,287
1/18/10	210,124	135,479
1/19/10	168,413	115,541

图6-16　原始的表格

我们可以很简单地把这个表格转换成图表，只需要把每一天放到恰当的位置上，然后伸展表格单元格到所需的高度，简单吧！

好吧，或许它没那么简单，不过你会发现它也没那么复杂。首先要设置工作区域：

```
table, table * {outline: 1px dotted red;}
table {display: block; position: relative;
    height: 300px; width: 600px;
font: small sans-serif;}
```

height和width可以使用任何度量单位，然而在接下来的工作中，像素可以使某些数学问题更容易解释。不过请放心，你要做的任何事都不会用到像素的。完成之后，你就可以随意把那些值改成你想要的值，而且仍然可以保持数据的图形化。outline的那条规则是临时的，它的存在只是让你看清楚你正在做什么。

现在开始为表格中的所有东西进行定位。

```
tr, th, td {display: block; position: absolute;}
tbody tr {left: 0; bottom: 0; width: 10%; height: 100%;}
```

所有tbody中的行都被设置为10%的宽度了，因为你希望它们肩并肩排列，而现在有10行，因此100%（表格的全部宽度）除以10就得到了10。然后只需把各行一个挨着一个排列（如图6-17所示）：

```
#day02 {left: 10%;}
#day03 {left: 20%;}
#day04 {left: 30%;}
```

```
#day05 {left: 40%;}
#day06 {left: 50%;}
#day07 {left: 60%;}
#day08 {left: 70%;}
#day09 {left: 80%;}
#day10 {left: 90%;}
tbody td {bottom: 0;}
```

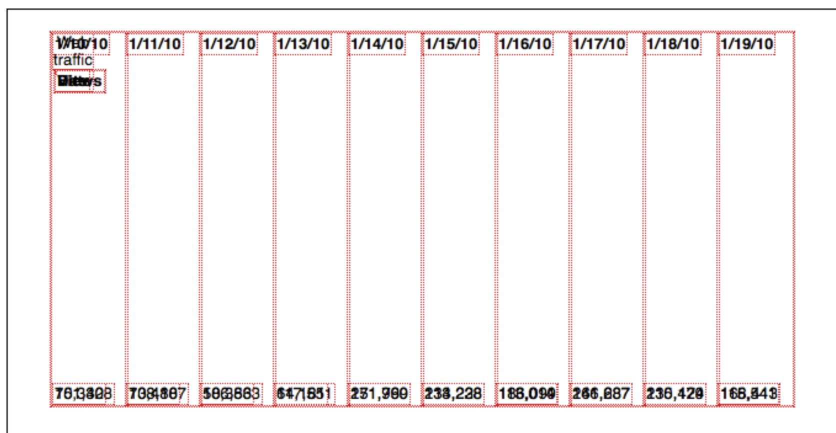


图6-17　为图表的每个条形设置容器

这是个有点儿令人乏味的部分：为每一行写出左侧的偏移量。想象一下整月都在干这个的情形！（当然，这就是为什么你会用脚本来写这种东西而不是手动去写的缘故。关于这一点，稍后我们会详细介绍。）

此时，该让每个条形恰当地“站”起来了。这里才是真正令人乏味的地方，因为需要为它们每一个单独计算高度然后再指定高度。首先需要确定最高的值，在本例中为251 969。然后选一个等于或高于这个值的数值作为图表的顶端。它可能是260 000、275 000或者甚至是300 000。对于这个例子，我们将使用260 000这个值。

完成这些之后，下一步就是用每个值除以刚才那个最大的值，从而得到一个百分比值。那么对于第一行，它包含了数字151 308和70 342，则对应的百分比值应为58.2%和27.05%。这些就是即将应用在第一行的td元素上的高度值。类似地，第二行的值为53.42%和27.08%。（是的，手工完成这些一定极为痛苦，用自动化的脚本可能更好一些。关于这一点，本章后续部分将会讨论到。）

```
#day01 td.hits {height: 58.2%;}
#day01 td.views {height: 27.05%;}
#day02 td.hits {height: 53.42%;}
#day02 td.views {height: 27.08%;}
```

对布局中的全部10行都使用同样的方式进行处理，结果如图6-18所示。

好了，已经有点儿阶梯状了，不过仍然很难看清条形。填充一些颜色会比较管用（如图6-19所示）。

```
.hits {background: #4444ED; color: #FFF;}
.views {background: #44ED44;}
```

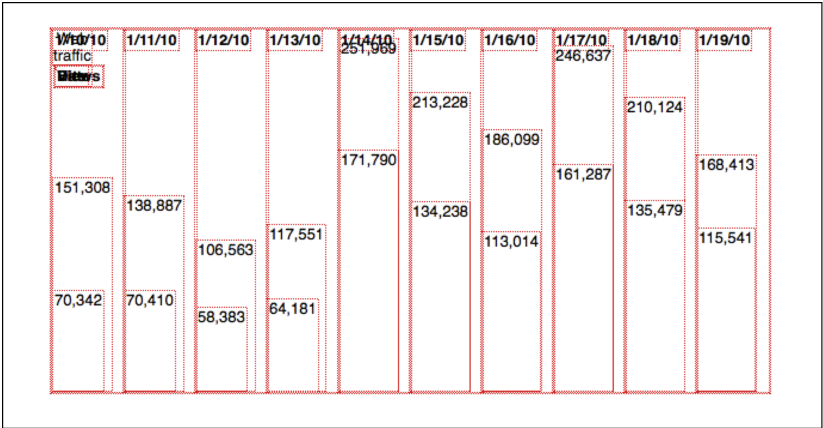


图6-18　设置条形的尺寸

这个简单的改变可以使条形变得更加明显。事实上，去掉点线的轮廓（红色）后就可以得到一个几乎足够理想的图了。这里有个关于thead和caption元素的很大问题：它们的内容仍然悬挂在左上角。此外还有一些小问题，比如图左侧的条形并非等宽，而且最高的1/14/10的条形覆盖了th的内容。

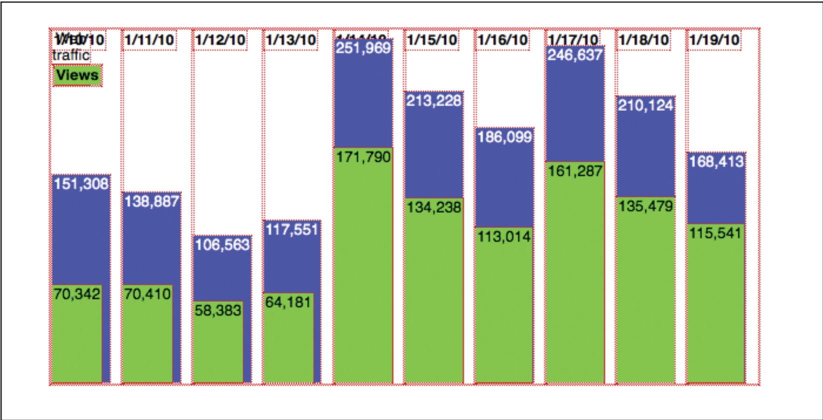


图6-19　添加颜色使条形更明显

我们将以逆序处理这些问题。首先，拿掉轮廓（红色）并且沿着整个图的顶部和底部分别设置一条细边框。

```
table, table * {outline: none;}
table {display: block; position: relative;
    height: 300px; width: 600px;
```

```
border: 1px solid #999; border-width: 1px 0;
font: small sans-serif;}
```

对于被覆盖的日期，把所有的日期一起移到表格的上方（如图6-20所示），以此来确保它们不会被覆盖。

```
tbody th {top: -1.33em;}
```

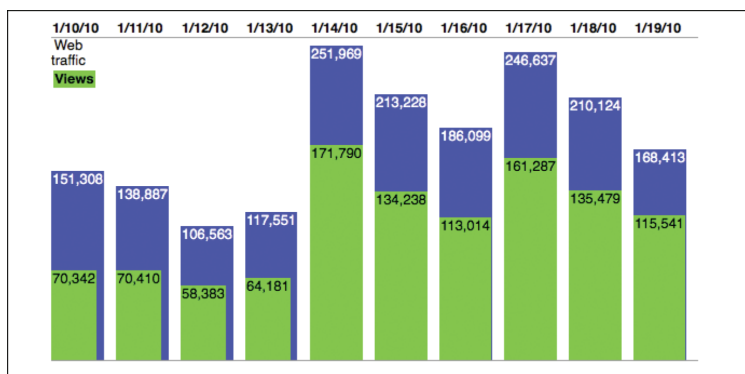


图6-20　清理表头的单元格

现在使所有的条形等宽，并使它们在其包含块（tr元素）中居中对齐。

```
tbody td {bottom: 0; width: 90%; left: 5%;}
```

事实上，文本可能看上去有一点儿错位，不过居中对齐可以使它看上去好一些（如图6-21所示）。如果你想让大部分内容都居中对齐的话，那么就在文档中更高的点进行设置。

```
table {display: block; position: relative;
    height: 300px; width: 600px;
    border: 1px solid #999; border-width: 1px 0;
    font: small sans-serif; text-align: center;}
```

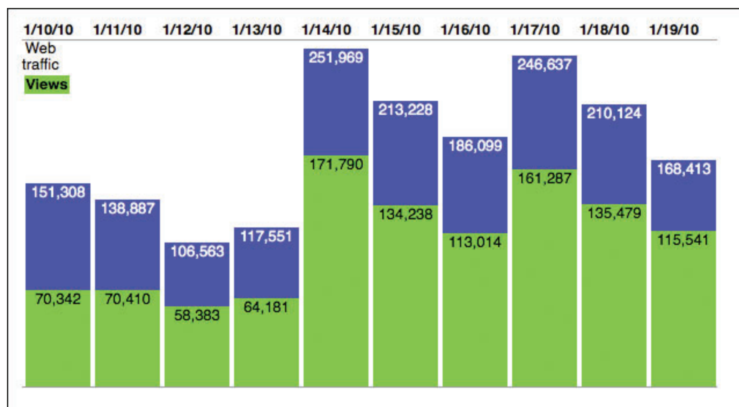


图6-21　使条形等宽

哎呀，日期没能跟条形中的文本一起对齐。这是因为你给td元素设置了宽度和左侧的偏移量，但是没有给th元素设置。那么可以改写一下那条规则，同时对作为tbody后代的th和td元素设置宽度和左侧偏移量——可以通过把tbody td这条规则中的width: 90%; left: 5%;移到它们自己的一条规则中来实现。

```
tbody th, tbody td {width: 90%; left: 5%;}
tbody td {bottom: 0;}
tbody th {top: -1.33em;}
```

那么还剩下什么呢？thead和caption，它们仍然乱堆在左上角。那么我们把caption放到表格的下方，设置为居中和粗体，就像这样：

```
caption {position: absolute; bottom: -1.75em; width: 100%;
    text-align: center; font-weight: bold;}
```

现在把thead变成这个图表的图例。毕竟，你需要的信息已经都在那儿了。

第一步就是去掉thead中对tr和th元素的定位。这个时候，由于有tr, th, td这条规则，因此它们是被绝对定位的。那么可以通过明确指定默认值static来覆盖那条规则，这基本上意味着“没有定位”。

```
thead * {position: static; padding: 0.25em;}
```

另外，你也可以调整tr, th, td这条规则的选择器，把它改成tbody tr, tbody th, tbody td。这样就不需要移除对thead后代元素的定位了，尽管你需要书写thead * {display: block;}使单元格堆叠在彼此之上。

对于本例中的这个表格，两种途径都能达到相同的效果（如图6-22所示）。那么，完成之后（完成任何一种方法）就该定位thead本身了。

```
thead {position: absolute;
top: 50%; margin-top: -2.5em;
left: 100%; margin-left: 2.5em;}
```

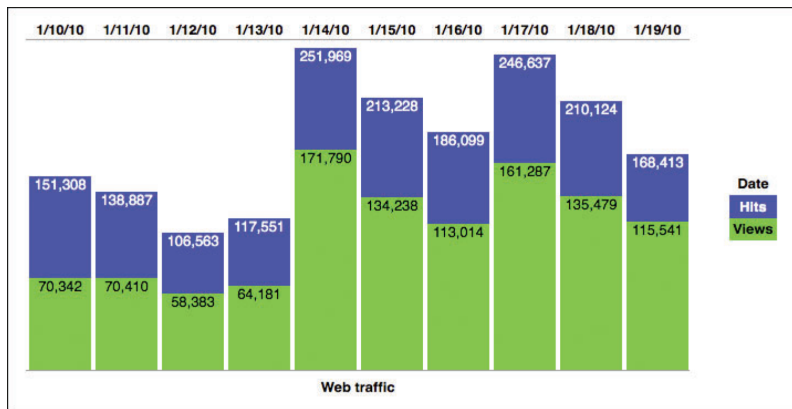


图6-22　使用列标题并将其作为图表的图例

大功告成了！

我许诺了好多次，说要考虑一下使用脚本完成一些操作过程，那么现在就开始吧。当计算图表的顶部边界（在这个特定的例子中是260 000），更别说是计算全部条形的高度时，很明显一点儿编程就有很大帮助，这可以通过下面两种方法实现。

第一种方法是在服务器端进行计算。在这种场景下，将要被图形化的数据是从数据库拉取出来的，而页面是通过某种模板生成的。这种情况下，你只需要把计算各种所需值的方法内建在模板中，也可能内建在它们各自的样式表中。

第二种方法是写一些JavaScript代码来完成这个繁重的任务。在这种方法中，各种条形的高度不会跟随图其他部分的样式一起被包含在样式表中。那么，一旦页面载入完成，JS将会遍历表格两次，一次用来收集所有的值并确定其中的最大值，一次用来分别对所有的`td`动态设置百分比高度。

第 7 章

可预见的未来

本章，我们关注那些即将到来的东西：那些可以立即使用或者在不远的未来就可以使用的应用样式的技术。从对HTML 5元素应用样式到基于显示参数重新安排布局，从疯狂的选择模式到转换元素布局，这些都是你在明天、下个月或者明年就可能用到的技术。尽管只有部分浏览器支持，但它们都是Web设计领域的前沿技术。

因此，一定要关注这些技术哦！这里有些网站可以帮助你确切地了解使用这些技术所需的相关语法以及模式。

- http://css3please.com/
- http://css3generator.com/
- http://www.westciv.com/tools/gradients/
- http://gradients.glrzad.com/

此外，还有一些JavaScript类库可以扩展老式浏览器对于高级CSS的支持，比如回溯至IE/Win 5.5时的一些情况。其中一些非常专注于某种特定的浏览器家族，而另外一些则希望更广泛地支持几乎所有已知的浏览器。如果访问者没能紧跟时代的步伐，而你又不希望他们错过所有好玩的地方，那这些就非常有用了。

- http://css3pie.com/
- http://www.useragentman.com/blog/csssandpaper-a-css3-javascript-library/
- http://www.keithclark.co.uk/labs/ie-css3/
- http://code.google.com/p/ie7-js/（实际上它远比这个URL展示的要强大得多。）
- http://ecsstender.com/

还有一些作为主流JavaScript类库的插件存在的CSS增强工具，比如jQuery的插件。如果你使用这种类库，那么一定要好好看看已经有什么可用的功能。再一次提醒，要小心！尽管这些技术很强大，而且可以为页面增添很多功能，但还是需要在各个浏览器中彻底地测试它们，以确保不会在一些老旧浏览器中使整个页面不可读。

7.1 为 HTML 5 应用样式

为HTML 5应用样式其实跟为HTML 4应用样式没什么区别。尽管有一大堆的新元素，但是

对它们应用样式与对任何其他元素应用样式基本上一模一样。它们与任何其他的div、span、二级标题、链接等元素生成的框完全相同。

截至撰写本文之时，HTML 5的规范仍然还在制定中，随着时间的推移这些可能会略有变化，不过下面的声明对于那些不是十分确定应该如何处理新元素的老旧浏览器可能会有用。

```
article, aside, canvas, details, embed, figcaption, figure, footer, header, hgroup,
 menu, nav, section, summary {display:block;}
command, datalist, keygen, mark, meter, progress, rp, rt, ruby, time, wbr {display:
 inline;}
```

你可能已经注意到了，我忽略了两个非常重要的元素：audio（音频）和video（视频）元素。那是因为我们很难确定应该如何对待它们。是块级？还是行级？这全都取决于你打算如何使用它们。总之，你可以把它们放到最有意义的声明中。

但是，对于那些真的非常非常老的浏览器（比如IE6），该怎么办呢？（注意我说的是“老的”，而不是“废弃的”，这是对流行文化的一个有趣颠覆，浏览器的流行程度跟它们的“年龄”无关。）对于这些元素，你需要使用一点儿JavaScript来使浏览器可以识别它们并对它们应用样式。这里有段很漂亮的脚本，参见http://remysharp.com/downloads/html5.js，它可以自动强制老版本的IE浏览器很好地兼容HTML 5元素。如果想使用它们并且对它们应用样式，那么你绝对应该使用这一脚本。

一旦准备好浏览器，就可以开始编写样式了。记住，为这些新元素应用样式与为之前的元素应用样式一模一样（如图7-1所示）。例如：

```
figure {float: left; border: 1px solid gray; padding: 0.25em; margin: 0 0 1.5em 1em;}
figcaption {text-align: center; font: italic 0.9em Georgia, "Times New Roman", Times,
serif;}

<figure>
    <img src="splash.jpg" alt="A toddler’s face is obscured by a rippled and
        dimpled wall of water thrown up by her hands slapping into the surface of
        the swimming pool in whose waters she sits.">
    <figcaption>SPLASH SPLASH SPLASH!!!</figcaption>
</figure>
```

图7-1　应用样式后的figure元素以及图标题

7.2　像 HTML 5 一样给类命名

或许你很喜欢HTML 5中的新语义，但可能还没准备好为站点全部采用HTML 5。可能你的站点用户大部分使用较老的浏览器，而且你也宁愿坚持去了解HTML 4或者XHTML的使用数量。那也不用担心，你可以通过备受推崇的`class`属性将两个世界很好地结合起来。

这种技术已由Jon Tan整理成文档并记录在他的文章中，文章地址为http://jontangerine.com/log/2008/03/preparing-for-html5-with-semantic-class-names。基本思想就是使用`div`和`span`等“老派”元素，并为它们添加与HTML 5元素的名称精确匹配的类名，下面是一个代码示例。图7-2展示了这个例子在浏览器中的渲染效果。

```
.figure {float: left; border: 1px solid gray; padding: 0.25em; margin: 0 0 1.5em 1em;}
.figcaption {text-align: center; font: italic 0.9em Georgia, "Times New Roman", Times,
serif;}

<div class="figure">
    <img src="spring.jpg" alt="A small child with twin pigtail braids,
        her back to the camera, swings away from the camera on a playground
        swingset while the late afternoon sun peeks over the crossbar of
        the swingset.">
    <div class="figcaption">Swinging into spring.</div>
</div>
```

图7-2　应用了`figure`和`figcaption`类的HTML 4元素

如果你把这个样式跟7.1节的样式进行对比，就会发现唯一的不同之处就是`figure`和`figcaption`这两个名字前面加了英文句点，因而它们变成了类名。当然，标记也有些不同，不过基本结构是一样的。

这种方法的优点就是，如果当你决定将站点转换成HTML 5站点时这些样式存在，那么只需用HTML 5元素替代原来添加了类名的`div`并去掉类选择器前面的句点，将其改为元素选择器。就是这样，小菜一碟儿！

7.3　媒体查询

这完全可以自成一章，甚至可能编撰成一本书。因此，下面的内容必然只是简单阐述一些可能性。你一定要跟进并多做些研究，因为在很多方面来看这都是Web设计的未来。

媒体查询（media query）的重点就在于设置可以应用在不同媒体环境的、分条件的样式代码块。例如，你可以写一组样式用于纵向显示内容，再写一组用于横向显示内容。你可以基于显示的位深改变颜色，也可以基于显示区域的像素密度更改字体，甚至可以依据显示区域的可用宽度或者可用的像素数量重新安排页面的布局（如图7-3所示）。

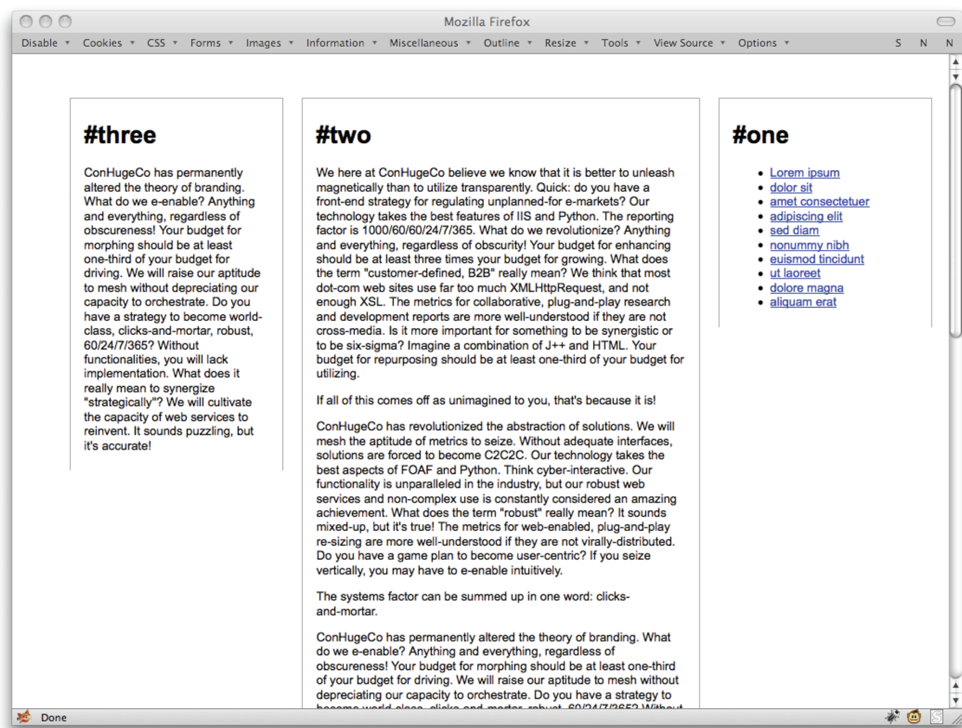


图7-3　基本的三栏布局

那么怎么实现呢？考虑一些用于三栏布局的基本样式。

```
body {background: #FFF; color: #000;
    font: small Arial, sans-serif;}
.col {position: relative;
    margin: 3em 1%; padding: 0.5em 1.5%;
    border: 1px solid #AAA; border-width: 1px 1px 0 1px;
float: right; width: 20%;}
#two {width: 40%;}
#footer {clear: both;}
```

它或许已经很好了（对那些要求很低的人来说），不过当它被缩小的时候很可能会遇到问题，我的意思是指当显示区域变窄的时候。那么如果在这种显示条件下，把它神奇地变为两列布局可不可以呢？

当然可以。首先，将三栏布局的应用环境限制为大于等于800 px的宽度。这可以通过分别将这些布局块放到它们自己的声明中来实现：

```
body {background: #FFF; color: #000;
    font: small Arial, sans-serif;}
.col {position: relative;
    margin: 3em 1%; padding: 0.5em 1.5%;
    border: 1px solid #AAA; border-width: 1px 1px 0 1px;}
#footer {clear: both;}
.col {float: right; width: 20%;}
#two {width: 40%;}
```

然后将最后两条规则用媒体查询包裹起来：

```
@media all and (min-width: 800px) {
    .col {float: right; width: 20%;}
    #two {width: 40%;}
}
```

它的意思就是“这个大括号中的规则将应用在显示宽度不小于800 px的全部媒体中”。不管是什么媒体，只要其显示宽度小于800 px，代码块中的规则就会被忽略。注意在`min-width`术语和值外面的括号，它在任何出现类似术语和值（它们也被称为表达式）的地方都是必需的。

此时，除非你把浏览器的窗口压缩到只能为文档提供小于800 px的宽度（如图7-4所示），否则不会看到任何变化。到那时，各栏就不会浮动了。

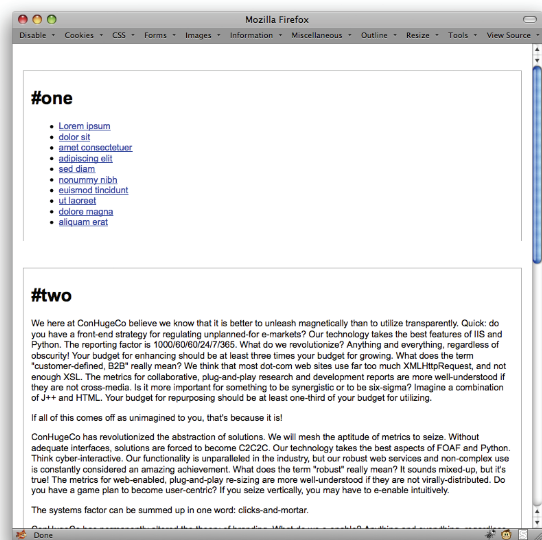


图7-4　小于800 px时的情景

此时你能做的就是另外再写一个可以应用在较窄条件下的媒体查询代码块。在此假设你想弄一个适用于500~800 px的两栏布局，如图7-5所示。

```
@media all and (min-width: 500px) and (max-width: 799px) {
    .col {float: left; width: 20%;}
    #two {float: right; width: 69%;}
    #three {clear: left; margin-top: 0;}
}
```

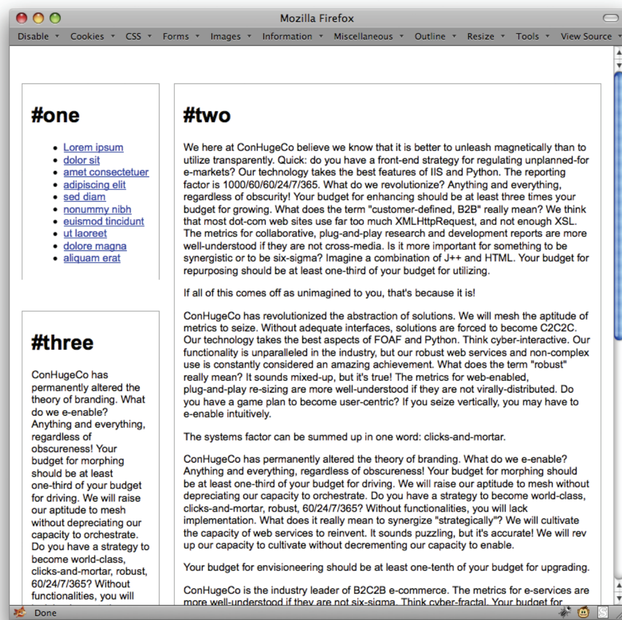


图7-5　在500~800 px宽度中显示调整后的布局

最后，可以对任何显示宽度小于500 px的媒体应用一些单栏的布局样式（如图7-6所示）。

```
@media all and (max-width: 499px) {
    #one {text-align: center;}
    #one li {display: inline; list-style: none;
        padding: 0 0.5em;
        border-right: 1px solid gray;
        line-height: 1.66;}
    #one li:last-child {border-right: 0;}
    #three {display: none;}
}
```

注意在所有的这些查询中，布局样式的定义都是跟浏览器窗口的显示区域相关联的。更一般地，它们是根据任何媒体为文档渲染所提供的可用显示区域的大小来定义的。这就意味着，如果一个打印机拥有784 px宽的可用显示区域，那么将会打印出两栏的布局。

为了限制各栏的移动只对屏幕媒体有效，应这样修改一下查询：

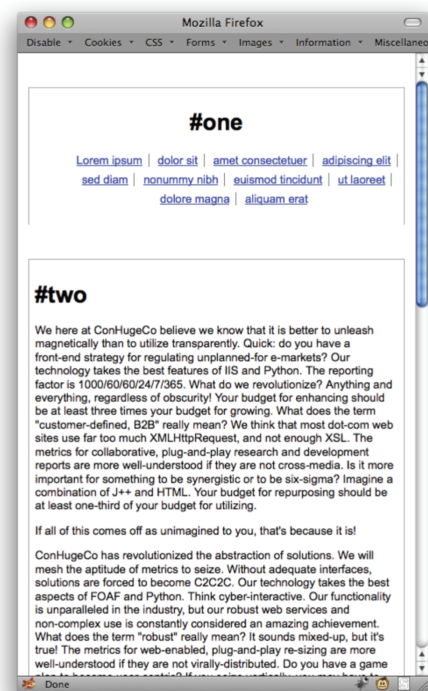


图7-6　宽度小于500 px时显示的单栏布局

```
@media screen and (min-width: 800px) {...}
@media screen and (min-width: 500px) and (max-width: 799px) {...}
@media screen and (max-width: 499px) {...}
```

但是假如你想让三栏布局可用在某些非屏幕媒体中，比如打印或者电视的显示上，该怎么办呢？那么加上这些媒体并使用逗号分隔即可，像这样：

```
@media print, tv, screen and (min-width: 800px) {...}
@media screen and (min-width: 500px) and (max-width: 799px) {...}
@media screen and (max-width: 499px) {...}
```

这里逗号的作用类似于逻辑或（OR），因此第一个查询可以理解为“在打印媒体或电视媒体**或**显示区域大于等于800 px的屏幕媒体上使用这些样式”。

那么，如果你想让三栏布局可应用在所有非屏幕媒体上呢？在第一个查询上使用not修饰符添加一个语句，声明“任何不是屏幕的媒体”。

```
@media not screen, screen and (min-width: 800px) {...}
@media screen and (min-width: 500px) and (max-width: 799px) {...}
@media screen and (max-width: 499px) {...}
```

像前面一样，逗号使两个语句成为逻辑或的关系，因此可以理解为“任何不在屏幕上的媒体或在屏幕上且显示区域大于等于800 px的媒体”。

这里还有个only修饰符，所以一个查询可以写成像only print或者only screen and (color)的样子。截至撰写本文之时，not和only是媒体查询仅有的两个修饰符。

顺便说一下，前面讲过的查询都不局限于使用像素。你可以使用em、厘米或者任何其他有效的长度单位。

表7-1展示了所有可以用来构建媒体查询的查询术语。注意，几乎所有这些术语都接受min和max前缀（例如，device-height同时有min-device-height和max-device-height两个“表亲”），只有orientation（方向）、scan（扫描）和grid（网格）例外。

表7-1 基本的媒体查询术语

术　语	描　述
width	显示区域的宽度（例如浏览器窗口宽度）
height	显示区域的高度（例如浏览器窗口高度）
device-width	设备显示区域的宽度（例如桌面显示器或移动设备的显示区域）
device-height	设备显示区域的高度
orientation	显示的方向，有两个值，即portrait（纵向）和landscape（横向）
aspect-ratio	显示区域的宽高比，值为斜线分割的两个整数
device-aspect-ratio	设备显示的宽高比，值为斜线分割的两个整数
color	显示设备的色彩位深，值为无单位的整数，代表位深。如果没有指定任何值，则会匹配任意色彩显示
color-index	设备的“色彩搜寻列表”（color lookup table）所维护的色彩数量，值为无单位整数
monochrome	应用在单色（或灰度）设备上
resolution	设备显示的分辨率，值以dpi或dpcm为单位
scan	“电视”媒体设备的扫描类型，值为progressive（逐行扫描）和interlace（隔行扫描）
grid	设备（例如一个TTY[①]设备）是否使用网格显示，值为0和1

7.4 为特定的子元素应用样式

有时，你可能需要选择某个序列中的第二、第三、第五、第八或者第十三个元素。最明显的例子就是一个很长的列表中的列表项，或者表格中的多行（或多列），不过这里的情况同元素的组合一样多。

考虑一种不太明显的情况，假设有一大堆引用自他人的句子，你希望把它们浮动到某种网格中。这些情况下通常会遇到的问题就是，引用的句子的不同长度可能打散网格，如图7-7所示。

① 原意为电传打字机（teletypewriter），现多指文本显示设备，如Linux终端等。

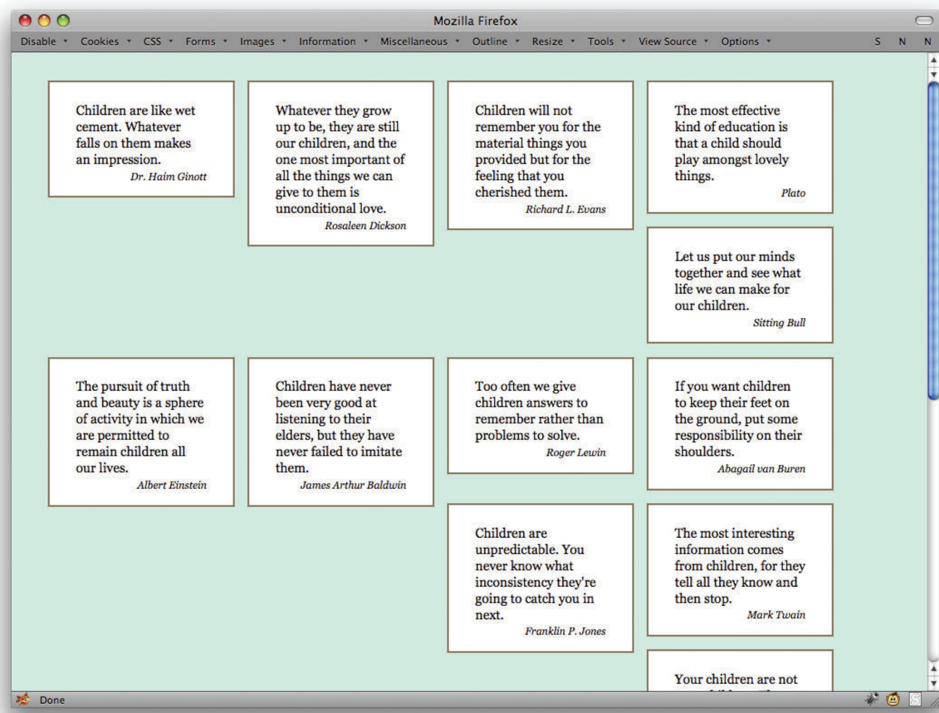


图7-7　变长元素浮动时所产生的问题

这里的一个经典解决办法就是，每隔4个[1]div（因为它是包含每个引用句子的元素）添加一个类然后清除它。然而，与其用类弄乱标记，为何不直接选择每隔4个的div呢（如图7-8所示）？

```
.quotebox:nth-child(4n+1) {clear: left;}
```

快速解释一下4n+1的部分。

- 4n意味着每个可以用公式4乘*n*来描述的元素，而*n*是指序列0,1,2,3,4…，从而得到元素的序号为0,4,8,12,16…（类似地，3n会得到0,3,6,9,12…）。
- 但是这里还包含零级的元素，元素应该是从第一个开始的（也就是，元素序号为1）。那么你需要增添+1表达式，才能选择第一、第五、第九个以及所有以此类推每隔4个的元素。

是的，你的理解很正确：那个:nth-child()的模式是从0开始计算的，而元素是从1开始计算的。这就是大多数:nth-child()选择器都会包含+1的缘故了。

使用这种选择器最棒的地方就是，如果不想选择每隔4个的元素，而想选择每隔3个的那些元素（如图7-9所示），那么修改一个数字即可。

```
.quotebox:nth-child(3n+1) {clear: left;}
```

① 这里的每隔4个是指取第1个、第5个、第9个，依次类推。

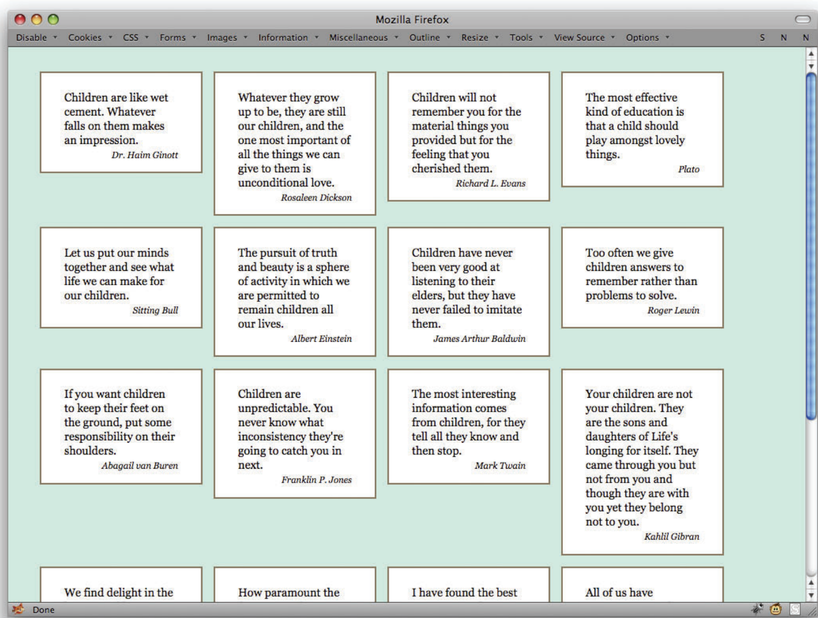


图7-8 清除每隔4个的子元素

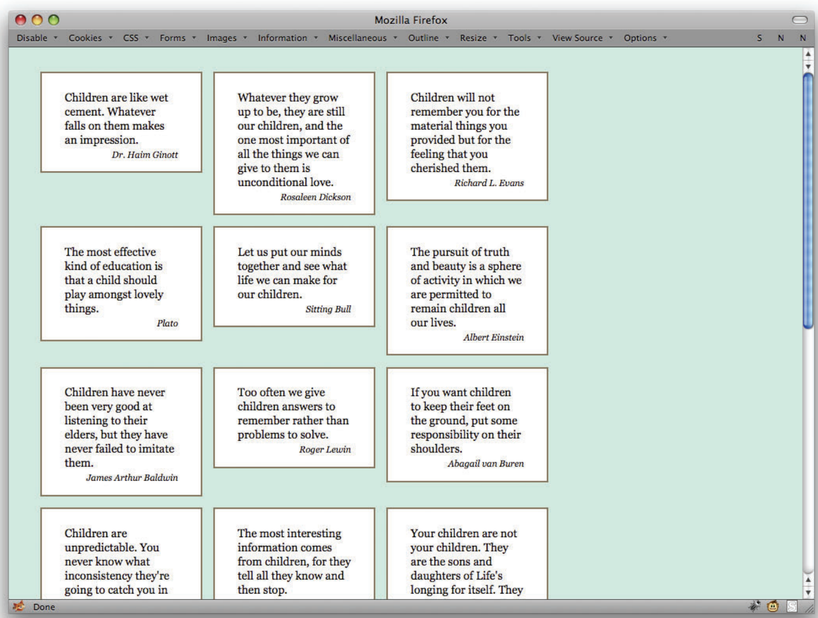


图7-9 清除每隔3个的子元素

可能它本身看上去已经非常绝妙了，不过它其实还能做得更好。如果你把这些方法与媒体查询结合起来，就可以得到一个适应性很好的网格状布局（见图7-10）。

```
@media all and (min-width: 75.51em) {
    .quotebox:nth-child(5n+1) {clear: left;}
}
@media all and (min-width: 60.01em) and (max-width: 75em) {
    .quotebox:nth-child(4n+1) {clear: left;}
}
@media all and (min-width: 45.51em) and (max-width: 60em) {
    .quotebox:nth-child(3n+1) {clear: left;}
}
@media all and (min-width: 30.01em) and (max-width: 45.5em) {
    .quotebox:nth-child(2n+1) {clear: left;}
}
@media all and (max-width: 30em) {
    .quotebox {float: none;}
}
```

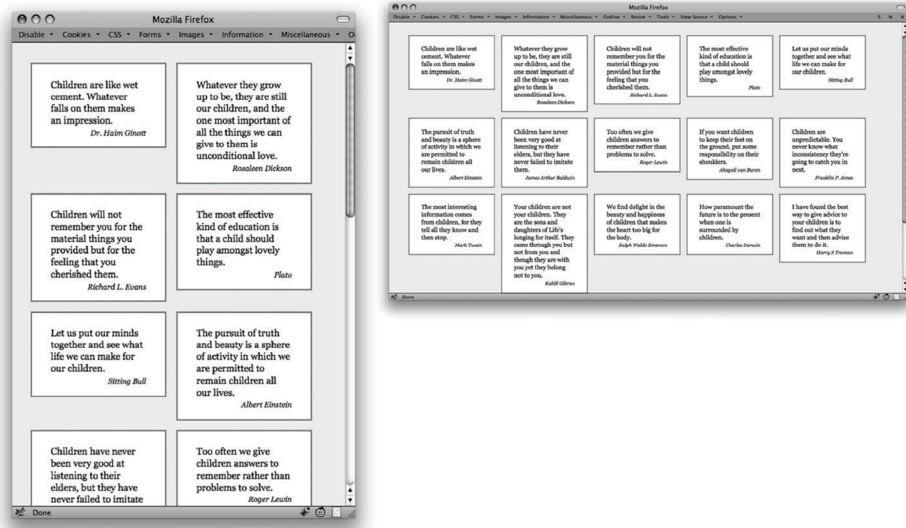


图7-10　自适应浮动网格的两种视图

注意，这组查询是基于以em为度量单位的浏览器显示区域的宽度确定的。这使得该布局在字号以及浏览器窗口改变时具有更好的适应性。

如果你还有兴趣选择每隔1个的元素，比如表格的隔行选择，那么这里有一些比`2n+1`更加人性化的替代方案。你可以通过`:nth-child(even)`和`:nth-child(odd)`选择偶数或者奇数的子元素。

```
tr:nth-child(odd) {background: #EEF;}
```

7.5　为特定的列应用样式

选取表格中交替的行并对其应用样式并非难事，但如果是列呢？实际上，也一样简单，因为我们有:nth-child和:nth-of-type选择器。

在一个每行仅由数据单元格（td元素）组成的简单表格中，你可以像这样进行隔列选择（如图7-11所示）：

```
td:nth-child(odd) {background: #FED;}
```

Jan	Feb	Mar	Apr	May	Jun	Jul
$11,940	$12,348	$14,301	$17,208	$16,087	$16,052	$16,404
$9,345	$9,834	$10,035	$9,672	$9,854	$9,405	$9,901
$2,787	$3,123	$4,137	$3,711	$3,092	$3,571	$2,811
$1,657	$3,003	$2,882	$2,690	$1,892	$2,292	$1,939
$8,947	$7,249	$8,102	$7,821	$7,654	$8,023	$8,197
$9,034	$11,027	$11,793	$10,283	$9,995	$10,562	$10,200
$10,633	$12,574	$12,834	$11,568	$12,130	$11,664	$11,243
$15,856	$16,239	$16,057	$15,712	$16,017	$15,784	$16,001
$8,245	$6,929	$6,498	$5,016	$6,909	$6,157	$5,047
$4,896	$4,869	$4,383	$6,808	$4,555	$6,619	$4,677
$83,340	$87,195	$91,022	$90,489	$88,185	$90,129	$86,420

图7-11　对奇数列应用样式（另见彩插图7-11）

还想像图7-12中那样填充间隔的那些列吗？超级简单！

```
td:nth-child(odd) {background: #FED;}
td:nth-child(even) {background: #DEF;}
```

如果想每隔3个（如图7-13所示）、4个、5个或者类似的区间间隔进行选择，那么就要用到n+1模式了。

```
td:nth-child(3n+1) {background: #EDF;}
```

Jan	Feb	Mar	Apr	May	Jun	Jul
$11,940	$12,348	$14,301	$17,208	$16,087	$16,052	$16,404
$9,345	$9,834	$10,035	$9,672	$9,854	$9,405	$9,901
$2,787	$3,123	$4,137	$3,711	$3,092	$3,571	$2,811
$1,657	$3,003	$2,882	$2,690	$1,892	$2,292	$1,939
$8,947	$7,249	$8,102	$7,821	$7,654	$8,023	$8,197
$9,034	$11,027	$11,793	$10,283	$9,995	$10,562	$10,200
$10,633	$12,574	$12,834	$11,568	$12,130	$11,664	$11,243
$15,856	$16,239	$16,057	$15,712	$16,017	$15,784	$16,001
$8,245	$6,929	$6,498	$5,016	$6,909	$6,157	$5,047
$4,896	$4,869	$4,383	$6,808	$4,555	$6,619	$4,677
$83,340	$87,195	$91,022	$90,489	$88,185	$90,129	$86,420

图7-12　同时对奇数列和偶数列应用样式（另见彩插图7-12）

Jan	Feb	Mar	Apr	May	Jun	Jul
$11,940	$12,348	$14,301	$17,208	$16,087	$16,052	$16,404
$9,345	$9,834	$10,035	$9,672	$9,854	$9,405	$9,901
$2,787	$3,123	$4,137	$3,711	$3,092	$3,571	$2,811
$1,657	$3,003	$2,882	$2,690	$1,892	$2,292	$1,939
$8,947	$7,249	$8,102	$7,821	$7,654	$8,023	$8,197
$9,034	$11,027	$11,793	$10,283	$9,995	$10,562	$10,200
$10,633	$12,574	$12,834	$11,568	$12,130	$11,664	$11,243
$15,856	$16,239	$16,057	$15,712	$16,017	$15,784	$16,001
$8,245	$6,929	$6,498	$5,016	$6,909	$6,157	$5,047
$4,896	$4,869	$4,383	$6,808	$4,555	$6,619	$4,677
$83,340	$87,195	$91,022	$90,489	$88,185	$90,129	$86,420

图7-13　对每隔3个的数据列应用样式（另见彩插图7-13）

这些相对来说都很简单。设想当把一个th放在每行开头时会发生什么呢？从某种意义上讲，什么也不会发生。选择的列不会有任何变化，你选择的仍然是tr元素的第一、第四、第七个等的子元素。而从另外的意义上讲，选择的列变动了，因为不再是第一、第四、第七个等的数据列了，而是第三、第六个等的数据列。由th元素组成的第一列根本不会被选择，原因是选择器只适用于td元素（如图7-14所示）。

	Jan	Feb	Mar	Apr	May	Jun	Jul
#207-B34	$11,940	$12,348	$14,301	$17,208	$16,087	$16,052	$16,404
#207-B35	$9,345	$9,834	$10,035	$9,672	$9,854	$9,405	$9,901
#207-B36	$2,787	$3,123	$4,137	$3,711	$3,092	$3,571	$2,811
#208-A07	$1,657	$3,003	$2,882	$2,690	$1,892	$2,292	$1,939
#208-A11	$8,947	$7,249	$8,102	$7,821	$7,654	$8,023	$8,197
#208-A12	$9,034	$11,027	$11,793	$10,283	$9,995	$10,562	$10,200
#208-A13	$10,633	$12,574	$12,834	$11,568	$12,130	$11,664	$11,243
#208-A23	$15,856	$16,239	$16,057	$15,712	$16,017	$15,784	$16,001
#209-C17	$8,245	$6,929	$6,498	$5,016	$6,909	$6,157	$5,047
#209-C55	$4,896	$4,869	$4,383	$6,808	$4,555	$6,619	$4,677
Total	$83,340	$87,195	$91,022	$90,489	$88,185	$90,129	$86,420

图7-14　用行标题扰乱刚才的选择模式（另见彩插图7-14）

为了修正这个问题，你可以修改:nth-child的表达式（如图7-15所示）：

```
td:nth-child(3n+2) {background: #EDF;}
```

	Jan	Feb	Mar	Apr	May	Jun	Jul
#207-B34	$11,940	$12,348	$14,301	$17,208	$16,087	$16,052	$16,404
#207-B35	$9,345	$9,834	$10,035	$9,672	$9,854	$9,405	$9,901
#207-B36	$2,787	$3,123	$4,137	$3,711	$3,092	$3,571	$2,811
#208-A07	$1,657	$3,003	$2,882	$2,690	$1,892	$2,292	$1,939
#208-A11	$8,947	$7,249	$8,102	$7,821	$7,654	$8,023	$8,197
#208-A12	$9,034	$11,027	$11,793	$10,283	$9,995	$10,562	$10,200
#208-A13	$10,633	$12,574	$12,834	$11,568	$12,130	$11,664	$11,243
#208-A23	$15,856	$16,239	$16,057	$15,712	$16,017	$15,784	$16,001
#209-C17	$8,245	$6,929	$6,498	$5,016	$6,909	$6,157	$5,047
#209-C55	$4,896	$4,869	$4,383	$6,808	$4,555	$6,619	$4,677
Total	$83,340	$87,195	$91,022	$90,489	$88,185	$90,129	$86,420

图7-15　通过调整选择方程式恢复选择模式（另见彩插图7-15）

另外，如图7-16所示，你也可以保留原来的模式，然后将:nth-child替换为:nth-of-type:

```
td:nth-of-type(3n+1) {background: #FDE;}
```

	Jan	Feb	Mar	Apr	May	Jun	Jul
#207-B34	$11,940	$12,348	$14,301	$17,208	$16,087	$16,052	$16,404
#207-B35	$9,345	$9,834	$10,035	$9,672	$9,854	$9,405	$9,901
#207-B36	$2,787	$3,123	$4,137	$3,711	$3,092	$3,571	$2,811
#208-A07	$1,657	$3,003	$2,882	$2,690	$1,892	$2,292	$1,939
#208-A11	$8,947	$7,249	$8,102	$7,821	$7,654	$8,023	$8,197
#208-A12	$9,034	$11,027	$11,793	$10,283	$9,995	$10,562	$10,200
#208-A13	$10,633	$12,574	$12,834	$11,568	$12,130	$11,664	$11,243
#208-A23	$15,856	$16,239	$16,057	$15,712	$16,017	$15,784	$16,001
#209-C17	$8,245	$6,929	$6,498	$5,016	$6,909	$6,157	$5,047
#209-C55	$4,896	$4,869	$4,383	$6,808	$4,555	$6,619	$4,677
Total	$83,340	$87,195	$91,022	$90,489	$88,185	$90,129	$86,420

图7-16 通过:nth-of-type恢复选择模式（另见彩插图7-16）

这之所以会起作用，是因为它选择了与其他元素共享同一个父元素的每个指定类型（本例中为td元素）的元素。你可以把它想象成一个:nth-child选择器，它可以跳过任何没有在:nth-child选择器中命名的元素。

7.6 RGB alpha 颜色

颜色值可能是所有CSS中最为人熟知的了，有些人甚至可以根据颜色的十六进制值估测该颜色的效果（用#E07713试试看）。尽管不像颜色的rgb()表示法那么常用，但它们确实十分受欢迎。

在CSS 3中，rgb()表示法被并入rgba()表示法了。值中的a代表alpha，即alpha通道透明度。因此你可以做出不完全透明的颜色（见图7-17所示）。

```
.box1 {background: rgb(255,255,255);}
.box2 {background: rgba(255,255,255,0.5);}
```

在RGBA中也可以使用百分比形式的RGB颜色值：

```
.box1 {background: rgb(100%,100%,100%);}
.box2 {background: rgba(100%,100%,100%,0.5);}
```

alpha值始终以一个在0和1闭区间内的数值来表示，其中0意味着“完全透明[①]”，而1意味着“完全不透明”。因此半不透明（half-opaque）是0.5，也即半透明度（half-transparent）是0.5。由于历史原因，这里不能使用百分比值，至于具体原因因为过于混乱就不便在此详述了。

如果提供一个在0和1闭区间之外的数值，它就会（规范中说的）被“钳制”在允许的范围内。因此，如果指定一个4.2的alpha值，浏览器就会当成你写的是1。同时，当alpha值为0时也不确定会发生什么情况。由于颜色是完全透明的，那会发生什么呢，看不见的文本？你能选择它吗？

① 请注意：Opacity的原意为“不透明度”，但中文中有时也译为“透明度”，因此很容易混淆。

如果是用在链接上的话，那链接可以单击吗？这些都是没有确切答案的有趣问题，所以要小心一点儿。

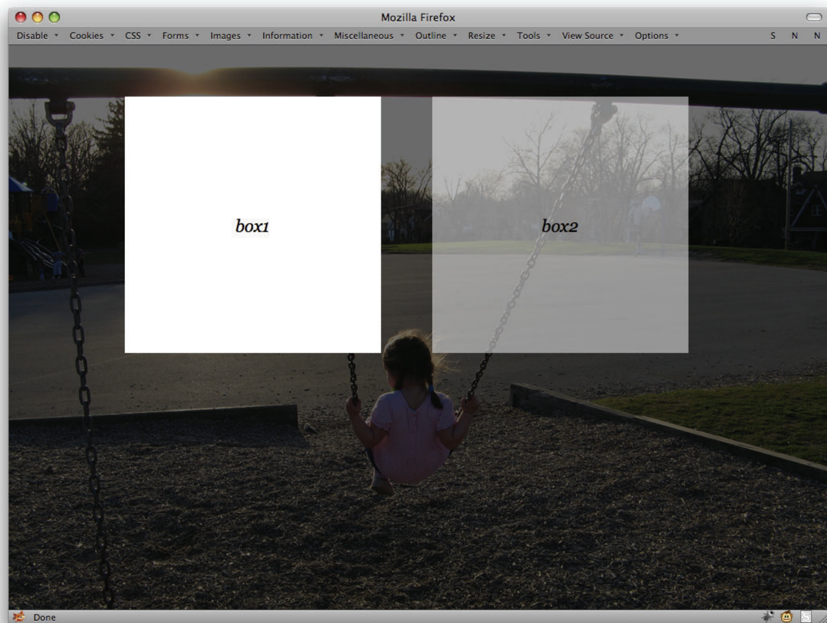


图7-17　不透明和半透明RGB背景的元素框

RGBA颜色可以用在任何能够接受颜色值的属性上，譬如color（字体颜色）和background-color（背景颜色）。为了兼容一些较老的浏览器，在alpha颜色前面增添一个非alpha颜色是个比较明智的选择，形式如下：

```
{color: #000; color: rgba(0,0,0,0.75);}
```

较老的浏览器可以看到第一个值并且知道如何处理这个值，然后它们会看到第二个值，但由于它们不知道如何处理这个值，因此会把它忽略掉。这样的话，至少较老的浏览器可以呈现黑色文本。另一方面，两个值都可以被现代浏览器识别，不过由于层叠的缘故，第二个值会覆盖第一个值。

注意，RGBA颜色没有十六进制值。因此，你不能指望通过写个#00000080得到半透明的黑色。

7.7　HSL 颜色和 HSL alpha 颜色

HSLA值与RGBA值非常相似，还有一种值与它更相似，就是HSL颜色。它们是CSS 3中新引入的，而且对于很多设计师来说将是一个很好的补充。

HSL表示Hue-Saturation-Lightness，即色调-饱和度-明度。即使不知道这个名字，可能你也曾在某个颜色拾取器中使用过HSL颜色，如图7-18所示。

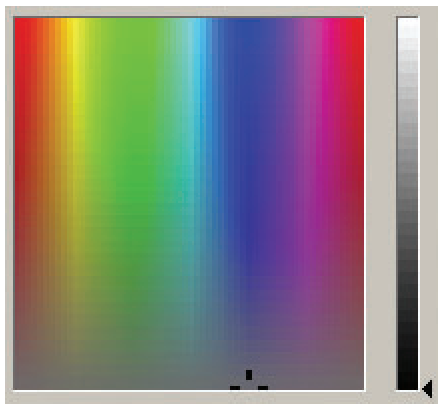


图7-18　HSL颜色拾取器（另见彩插图7-18）

图7-19展示了一些颜色表，可以让你大致了解HSL颜色的各部分是如何一起工作的。

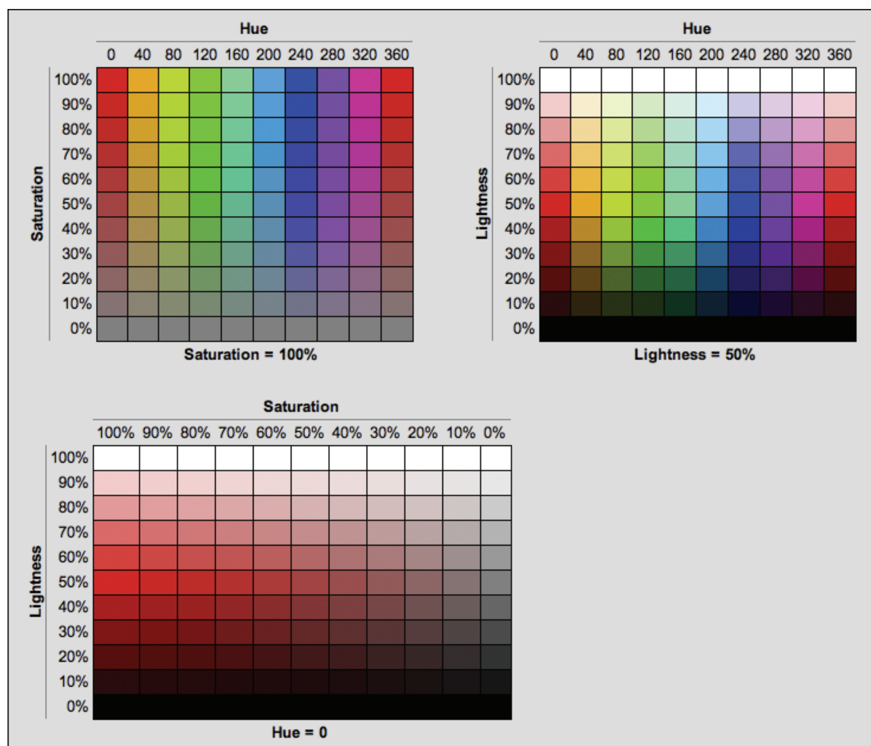


图7-19　各种HSL颜色表（另见彩插图7-19）

色调是用无单位的数值表示的，对应着色盘中的色调角度。饱和度和明度都是百分比，而alpha值（跟RGBA一样）为一个在0和1闭区间内的数值。在实际应用中，你可以在任何能接受颜色值的地方使用HSL颜色。考虑下面的规则，它可以创建跟图7-17相同的效果（如图7-20所示）。

```
.box1 {background: hsl(0,0%,100%);}
.box2 {background: hsla(0,0%,100%,0.5);}
```

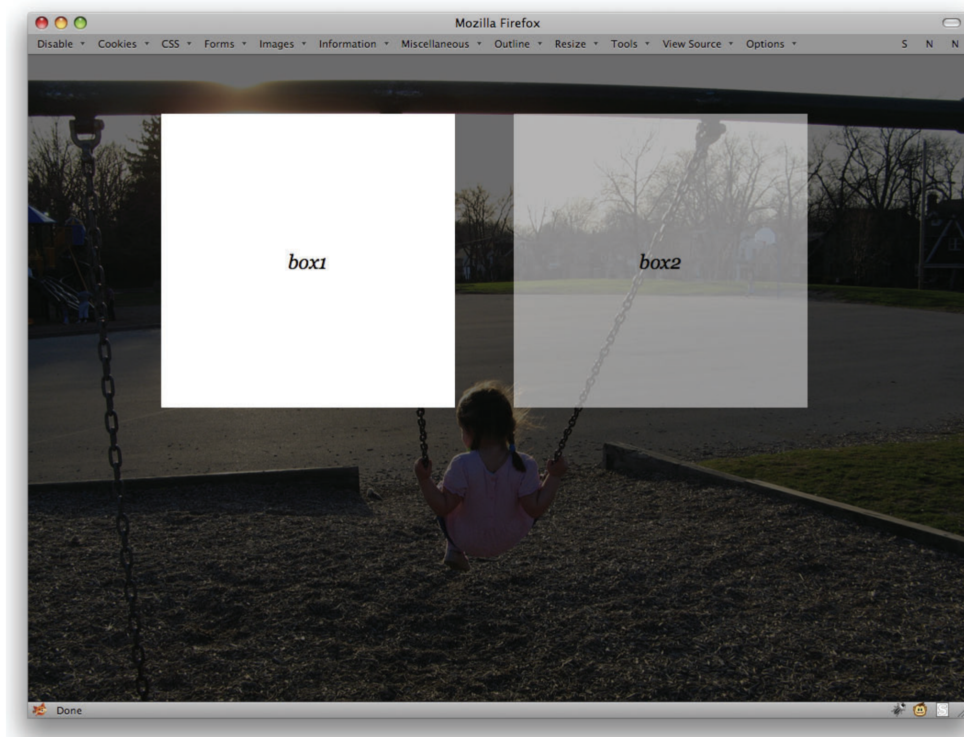


图7-20　不透明和HSL透明背景的元素框

你可以使用常规的RGB值对老式浏览器做些回退，然而先指定RGB颜色，然后再指定HSL颜色，相对于把HSL放在第一位来说会失色不少。HSL可以让你完全放弃RGB颜色。

7.8　阴影样式

啊，阴影！还记得阴影吗？在20世纪90年代中期，几乎啥都有阴影。当然，遥想当年阴影都是通过图像和表格构建的，甚至比通常的方法更加迂回曲折。现在你可以通过一些非常简单的CSS重新体验一下那个辉煌的时代了。

实际上有两个可用的属性：`text-shadow`和`box-shadow`。先说第一个，以下CSS的结果如图7-21所示。

Running Between the Shadows

图7-21　为标题设置阴影

```
h1 {text-shadow: gray 0.33em 0.25em 0.1em;}
```

第一个长度（0.33em）代表水平偏移量，第二个（0.25em）代表垂直偏移量。第三个是模糊半径，它代表了阴影被模糊的程度。这些值可以使用任何长度单位，因此如果你想用像素指定所有的阴影偏移量和模糊程度，那么尽管去做好了。模糊程度不能是负值，但是偏移量可以：负的水平偏移量会将阴影推向左侧，而负的垂直偏移量会把阴影向上拉。

如图7-22所示，你甚至可以设置多个阴影！当然，该不该这么做却是见仁见智的事情。

```
h1 {text-shadow: gray 0.33em 0.25em 0.1em, -10px 4px 7px blue;}
```

Running Between the Shadows

图7-22　拥有多个阴影的标题

注意，阴影的颜色值可以放在所有长度之前，也可以放在它们之后，这要看你的喜好了。同时还要注意，CSS 3规范里说第一个阴影是“在上面”的，即最接近你的。从你观看页面的视角来看，后面的阴影会依次地按照远离你的顺序放置。因此，灰色的阴影被放在了蓝色的阴影上面。

现在为框设置阴影（如图7-23所示）。几乎如出一辙，只是属性名不同而已。

```
h1 {box-shadow: gray 0.33em 0.25em 0.25em;}
```

Running Between the Shadows

图7-23　为标题的元素框设置阴影

虽然这个一级标题（h1）没有明显的框，还是生成了阴影。同时，阴影也仅在元素的外部绘制，这意味着你无法看到在元素后面/下面的阴影，即使元素拥有透明（或者RGBA颜色的半透明）的背景。阴影只能绘制在边框边缘之外，因此为任何设置了阴影的元素框设置一个边框或者一个可见的背景（或者两者同时设置）可能会比较好。

如图7-24所示，你也可以设置多个框阴影，就像添加文本阴影一样。

```
h1 {box-shadow: gray 0.33em 0.25em 0.25em, -10px 2px 6px blue;}
```

Running Between the Shadows

图7-24　标题的元素框上的多个阴影

这里我不得不承认一个小谎言：前面的例子是在理想情况下的效果。截至撰写本文之时，它们实际上无法在所有浏览器中良好工作。事实上，这些插图是使用跟上面展示的文本不同的语法做出来的。截至2010年年中，若想使单个阴影的例子正常工作，实际上需要这样写：

```
h1 {-moz-box-shadow: gray 0.33em 0.25em 0.25em;
       -webkit-box-shadow: gray 0.33em 0.25em 0.25em;
       box-shadow: gray 0.33em 0.25em 0.25em;}
```

这会兼容截至2010年年中时的所有现代浏览器。随着时间的推移，属性的这些前缀（-moz-和-webkit-）会慢慢消失，那时就可以只使用单独的box-shadow声明了。具体什么时候能实现呢？我想还是取决于你的设计、站点的访问者，还有你自己的舒适感了。

如果你想在较老版本的IE浏览器中也为元素框呈现阴影的话，那么需要添加IE专属的阴影滤镜。详见http://robertnyman.com/2010/03/16/drop-shadow-with-css-for-all-web-browsers/。

7.9　多背景

CSS 3中真正时髦的东西之一，就是它对于给定元素支持多个背景图像。如果曾经嵌套过多个div元素，只是为了做出一堆背景，那么本节正是为你准备的。

以这组简单的样式和标记为例，我们来展示一段引文，结果如图7-25所示。

```
body {background: #C0FFEE; font: 1em Georgia, serif; padding: 1em 5%;}
.quotebox {font-size: 195%; padding: 80px 80px 40px; width: 16em; margin: 2em auto;
border: 2px solid #8D7961; background: #FFF;}
.quotebox span {font-style: italic; font-size: smaller; display: block; margin-top:
0.5em; text-align: right;}
<div class="quotebox">
One&#8217;s mind has a way of making itself up in the background, and it suddenly
becomes clear what one means to do.
<span>&#8212;Arthur Christopher Benson</span>
</div>
```

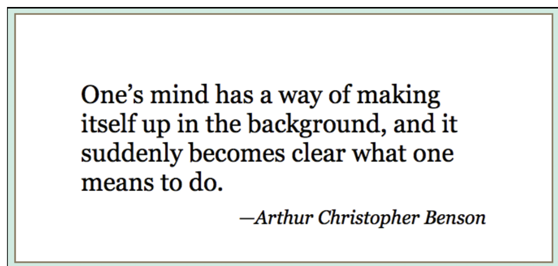


图7-25　设置引文框

现在，添加一个单独的背景图像（如图7-26所示）没有什么难度，每个人都这样做过无数次了。

```
.quotebox {background: url(bg01.png) top left no-repeat; background-color: #FFF;}
```

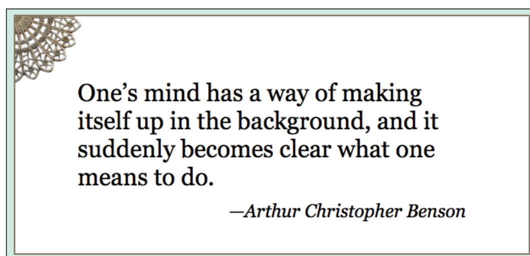


图7-26　添加单独的背景

但是如果想让每个角都显示1/4的轮子呢（如图7-27所示）？显然，你需要在quotebox这个div中嵌套一堆div。有了CSS 3，只需将它们添加到background（背景）声明中。

```
.quotebox {background:
             url(bg01.png) top left no-repeat,
             url(bg02.png) top right no-repeat;
      background-color: #FFF;}
```

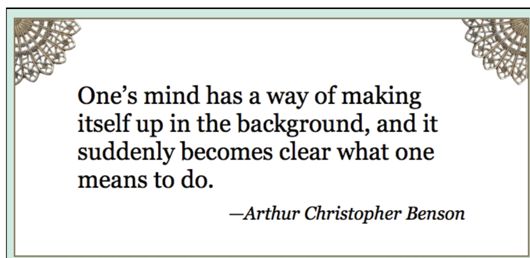


图7-27　对同一元素应用2个背景

以逗号分隔每个background值来获得多个背景（如图7-28所示）。

```
.quotebox {background:
             url(bg01.png) top left no-repeat,
             url(bg02.png) top right no-repeat,
             url(bg03.png) bottom right no-repeat,
             url(bg04.png) bottom left no-repeat;
      background-color: #FFF;}
```

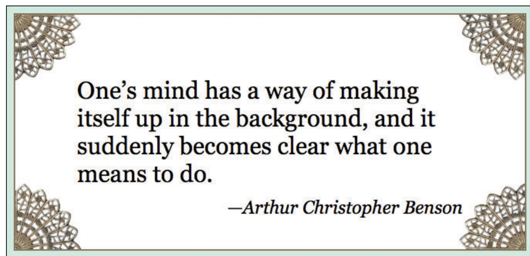


图7-28　对同一元素应用4个背景

这个效果跟应用一堆嵌套div的效果极其相似，只不过它是用CSS 3实现的，你不需要再纠结了。

这种相似性也存在于组合背景的方式中。你可能注意到我把background-color声明单独拿出来了，目的就是为了在所有图像下面能有个纯白色的背景。但是，如果你想把它放到background声明中呢？把它放在哪儿？毕竟，每一个逗号分隔的值都是在设置它自己的背景。如果把颜色放错了地方，那么就可能会有一个或多个图像被相应颜色覆盖。

事实证明，答案就是放在最后一个值中。

```
.quotebox {background:
             url(bg01.png) top left no-repeat,
             url(bg02.png) top right no-repeat,
             url(bg03.png) bottom right no-repeat,
             #FFF url(bg04.png) bottom left no-repeat;}
```

这是因为多背景的应用顺序是从“最高”到“最低”的，即以你观看页面的视角来看，是从最靠近你的到离你最远的顺序应用。如果你把颜色放到第一个背景上，它就会位于其他背景的“上面”。

这也意味着，如果你想让某种图案背景在所有背景的最后方（如图7-29所示），那么就需要把它放在最后一个并且确保把任何背景颜色的值也移到这个值中。

```
.quotebox {background:
             url(bg01.png) top left no-repeat,
             url(bg02.png) top right no-repeat,
             url(bg03.png) bottom right no-repeat,
             url(bg04.png) bottom left no-repeat,
             #FFF url(bgparch.png) center repeat;}
```

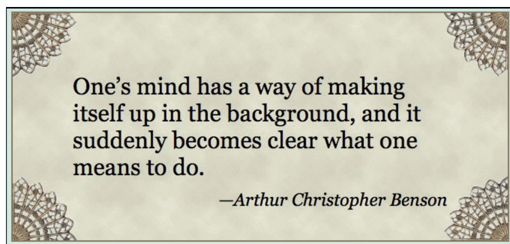


图7-29　1个元素，5个背景

由于可能引入的复杂性，我比较喜欢把任何默认的背景颜色放到它们自己的声明中，就像之前展示的那样。因此，我会把前面的样式写成这样：

```
.quotebox {background:
             url(bg01.png) top left no-repeat,
             url(bg02.png) top right no-repeat,
             url(bg03.png) bottom right no-repeat,
             url(bg04.png) bottom left no-repeat,
             url(bgparch.png) center repeat;
      background-color: #FFF;}
```

当使用分隔的属性时，把颜色放到所有图像的后面，这样在重新调整图像的顺序或者增添新图像时就不用再把它们挪来挪去了。

你也可以用逗号分隔其他的背景属性，如background-image。事实上，前面的样式还有另一种写法：

```
.quotebox {
    background-repeat: no-repeat, no-repeat, no-repeat, no-repeat, repeat;
    background-image: url(bg01.png), url(bg02.png), url(bg03.png), url(bg04.png),
url(bgparch.png);
    background-position: top left, top right, bottom right, bottom left, center;
    background-color: #FFF;}
```

不同的格式，同样的效果。这种格式可能看起来很累赘，而且在这种情况下也确实是，但并不总是这样。如果你放弃羊皮纸的背景（结果如图7-30所示），那么就可以简化一下第一个声明了：

```
.quotebox {
    background-repeat: no-repeat;
    background-image: url(bg01.png), url(bg02.png), url(bg03.png), url(bg04.png);
    background-position: top left, top right, bottom right, bottom left;
    background-color: #FFF;}
```

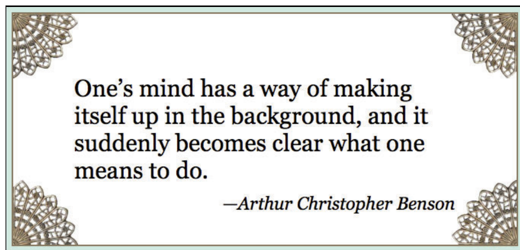


图7-30　类似的背景，替代的语法

给定这些样式后，就不会有任何背景被平铺了，因为那个no-repeat会应用到元素的所有背景上。你需要写出所有平铺值的唯一理由就是，前4个是一个值，而第五个是另外一个值。

如果你写出两个background-repeat值会怎样呢？

```
.quotebox {
    background-repeat: no-repeat, repeat-y;
    background-image: url(bg01.png), url(bg02.png), url(bg03.png), url(bg04.png);
    background-position: top left, top right, bottom right, bottom left;
    background-color: #FFF;}
```

在那样的情况下，第一个和第三个图像不会被平铺，而第二个和第四个会被沿着*Y*轴平铺。如果是3个平铺值的话，它们会分别被应用到第一、第二和第三个图像上，而第四个图像会取第一个平铺值。

7.10　二维变换

如果你曾想过旋转或倾斜一个元素、边框、文本或其他任何东西，那么这一节绝对适合你。

然而，首先要提醒一句：为了保持易读性，本节将使用transform（变换）属性的无前缀版本。截至撰写本文之时，在浏览器中实现变换实际上需要多个带前缀的声明，就像这样：

```
-webkit-transform: …;
-moz-transform: …;
-o-transform: …;
-ms-transform: …;
transform: …;
```

在一两年之内应该就不需要了（希望如此！），不过与此同时，当阅读本节时要时刻记着，这里只是为了清晰起见而使用的无前缀简化版本。

到了变换的时候啦！可能最简单最容易理解的变换就是旋转了，如图7-31所示。（在下一个及后面的图中，虚线（实为红色）代表被变换的元素在变换之前的放置位置。）

```
.box1 {-moz-transform: rotate(33.3deg);}
.box2 {-moz-transform: rotate(-90deg);}
```

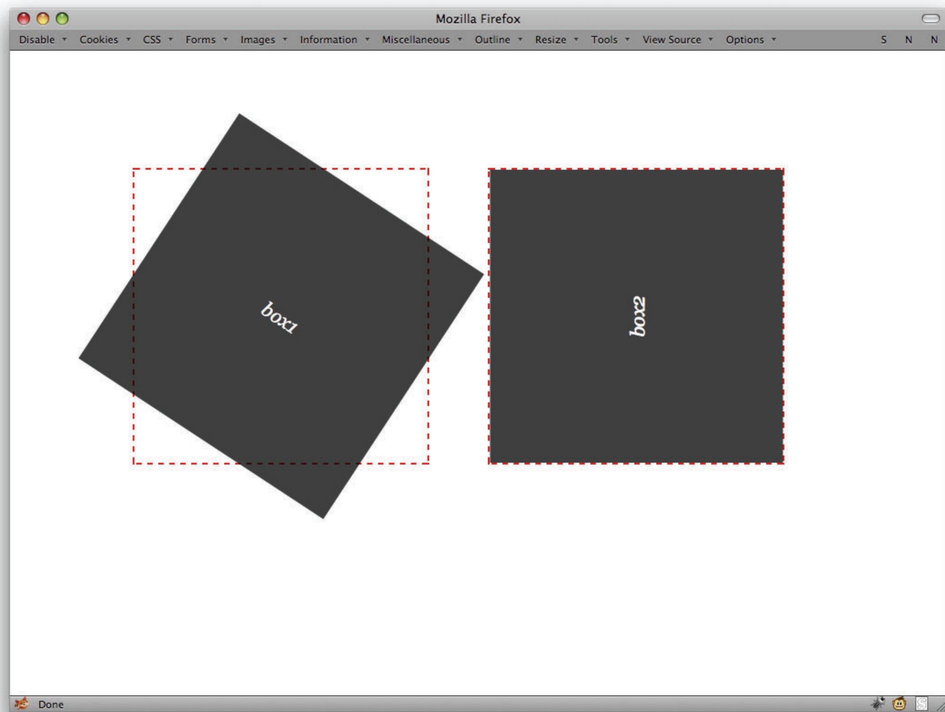


图7-31　旋转后的元素框，虚线展示了元素在被旋转之前的原始放置位置

在某种意义上，变换跟相对定位很像：元素被正常放置[1]，然后进行变换。你可以变换任何元素，在使用旋转时可以应用任何实数的度数（degree）、弧度（radian）或者百分度（grad）来指定旋转角度。如果你曾想过把博客旋转e弧度或者225百分度[2]，那么好吧，你的机会来了。

毫无疑问你肯定注意到了，前面例子中的元素框是围绕它们的中心旋转的。这是因为默认的变换原点是`50% 50%`，或者说是元素的中心。你可以通过`transform-origin`改变原点（如图7-32所示）。

```
.box1 {transform: rotate(33.3deg); transform-origin: bottom left;}
.box2 {transform: rotate(-90deg); transform-origin: 75% 0;}
```

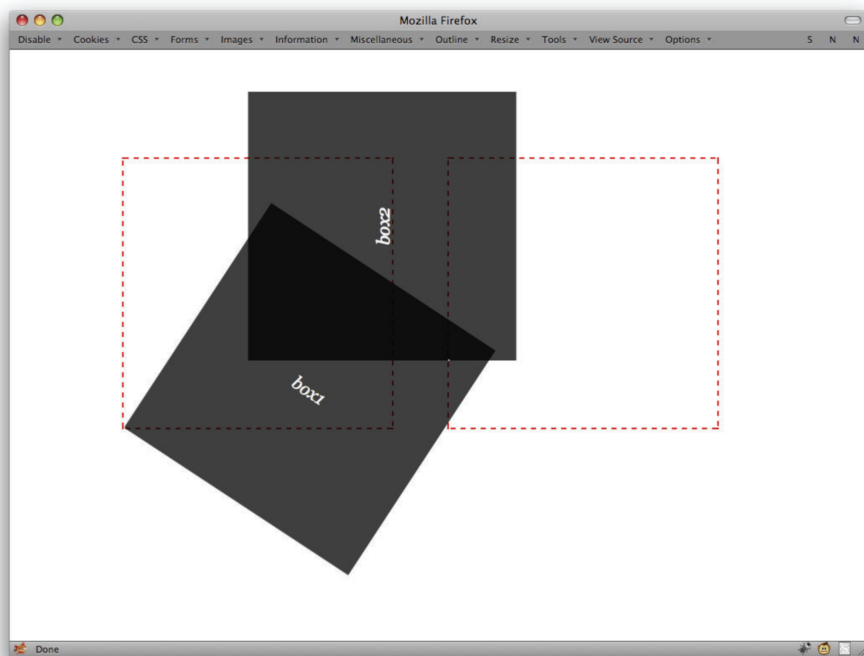


图7-32　元素围绕非中心的某个点旋转

两点需要注意。第一点，负的角度可以用等价的正角度实现。因此，270 deg[3]与–90 deg在元素的最终位置上是等价的，就像0 deg和360 deg是等价的一样。第二点，你可以指定比表面看上去的最大值更大的角度。如果你声明540 deg，则元素旋转后的最终位置和声明180 deg的效果是完全相同的（也和–180 deg、900 deg等相同）。如果你同时应用了变换（见下一节）的话，那么中间的结果可能会不同，不过最终“静止”的状态是等价的。

① 这里指元素位于正常的文档流中，相当于`position:static`。

② 1弧度约为57.3度，e约等于2.718 28，1个百分度为直角的1%，即0.9度。

③ 即270 degreee和270度，下同。

跟旋转一样简单的还有缩放（scale），图7-33中展示了一个例子。正如你所期待的，它可以缩放元素的尺寸，使元素变小或者变大。你可以沿着两个坐标轴，或者以某个角度沿着每个坐标轴进行缩放。

```
.box1 {transform: scale(0.5);}
.box2 {transform: scale(0.75, 1.5);}
```

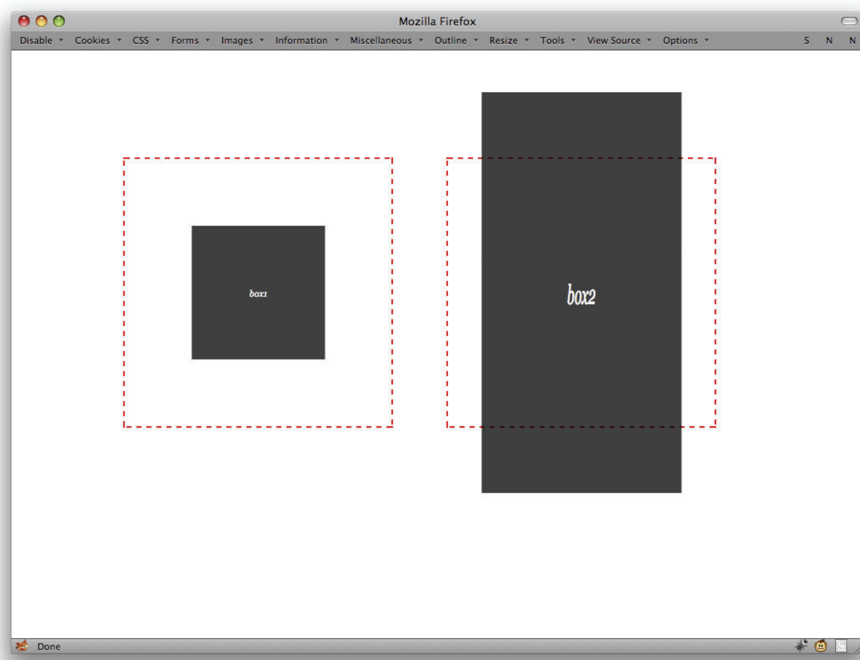


图7-33　缩放后的元素

一个scale()值意味着元素会同时沿着X轴和Y轴缩放那些倍数。如果有两个值，则第一个代表水平（X）缩放，第二个代表垂直（Y）缩放。因此，如果你想保持水平方向不变，而只在Y轴进行缩放，那就这样做：

```
.box1 {transform: scale(0.5);}
.box2 {transform: scale(1, 1.5);}
```

另外，也可以使用scaleY()值：

```
.box1 {transform: scale(0.5);}
.box2 {transform: scaleY(1.5);}
```

无论你选择了哪种特定的方式，图7-34都将是最终结果。

对于scaleX()的值也是一样，它会导致垂直方向不变，而只缩放水平方向（如图7-35所示）。

```
.box1 {transform: scaleX(0.5);}
.box2 {transform: scaleX(1.5);}
```

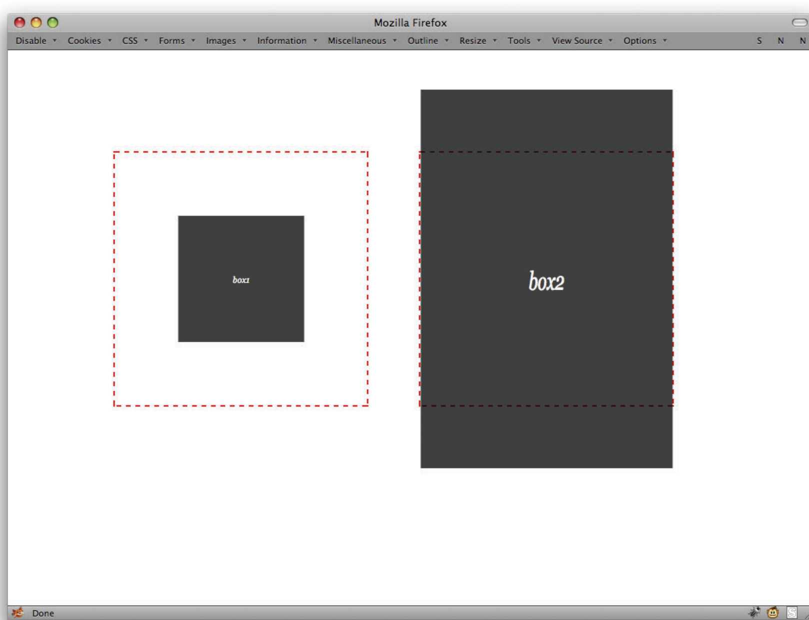


图7-34　两个缩放后的元素，其中一个只沿*Y*轴缩放

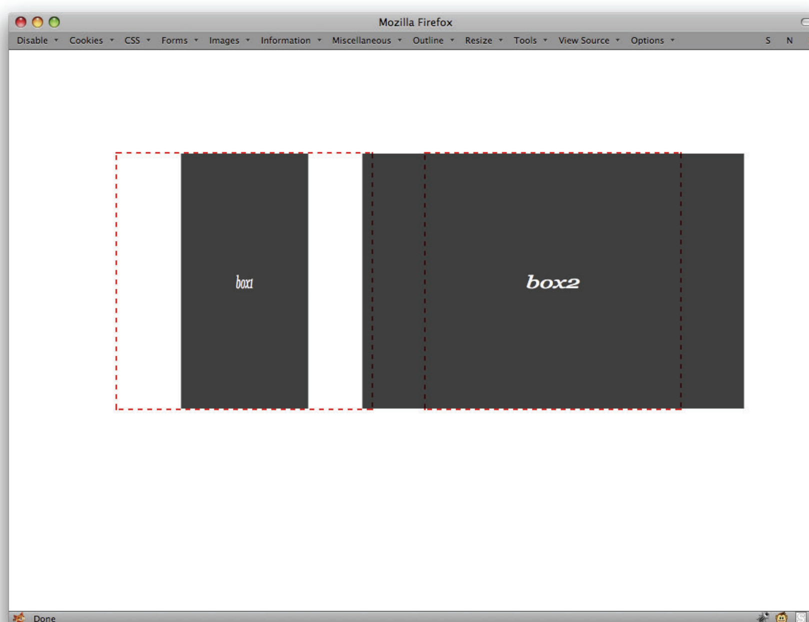


图7-35　两个沿*X*轴缩放后的元素

自己写CSS的时候，在任何需要纯垂直方向缩放的情况下，只使用scale()并填入一个0的方式看起来是最方便的。如果通过DOM脚本编程来改变元素的缩放，那么直接操作scaleX()和scaleY()可能更容易些。

像旋转一样，你也可以影响缩放的原点。例如，可以使一个元素沿着其左上角进行缩放，而不是沿着中心收缩（如图7-36所示）。

```
.box1 {transform: scale(0.5); transform-origin: top left;}
.box2 {transform: scale(1.5); transform-origin: 100% 100%;}
```

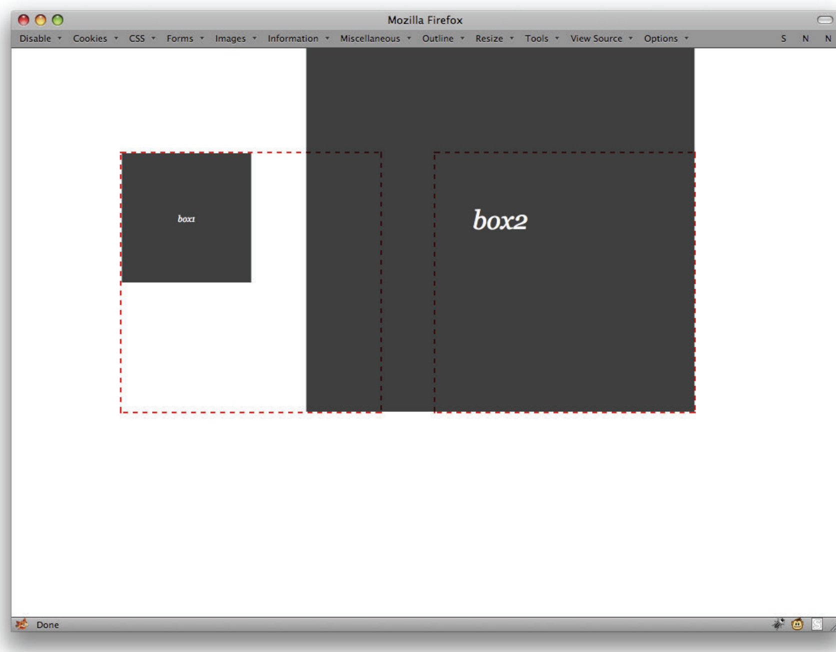


图7-36　两个拥有不同缩放原点的元素（已缩放）

同样简单的还有平移。在这种情况下，它不止是换了种说法，而是将一个形状从一点“平移”到另外一点，如图7-37所示。它是包含了一个或两个长度值的偏移量。

```
.box1 {transform: translate(50px);}
.box2 {transform: translate(5em,10em);}
```

再一次的，它也跟相对定位非常像。这些元素被正常放置，然后直接进行平移。

当translate()值中只有一个长度值时，它指定水平的移动量，而垂直的移动量则假定为零。如果你只想让元素向上或者向下平移，那么有两个选择。第一个，简单地为水平方向指定一个为0的长度值。

```
.box1 {transform: translate(0,50px);}
.box2 {transform: translate(5em,10em);}
```

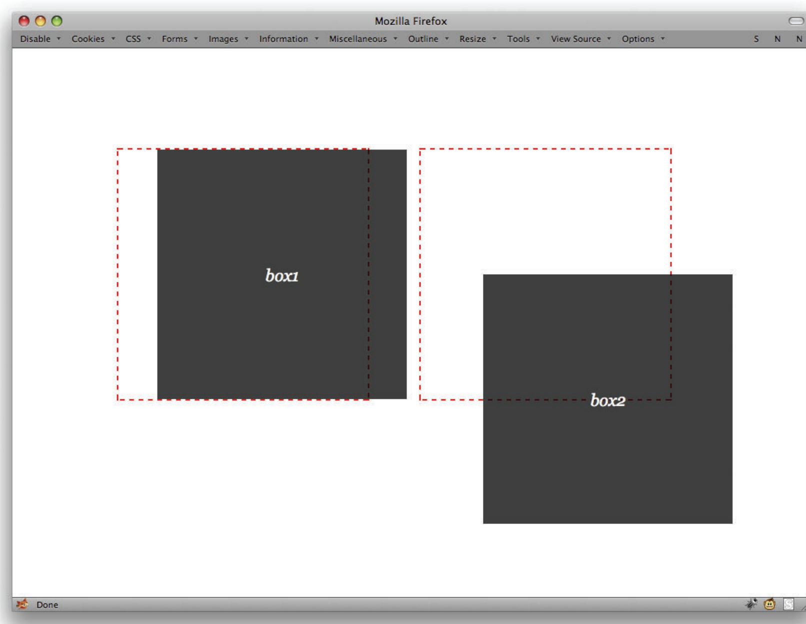


图7-37　平移后的元素

另外一个是使用translateY()的值模式：

```
.box1 {transform: translateY(50px);}
.box2 {transform: translate(5em,10em);}
```

任何一种方法都会得到如图7-38所示的结果。

这里同样存在translateX()，它也会像你预期的那样工作：水平移动元素。

你可以在刚刚平移的地方再声明一个transform-origin，当然是否这么做并不是很重要。毕竟，到底是元素的中心还是左上角被向右推过50 px都无所谓，无论哪种方式元素的最终位置都是一样的，不过这只在你使用的全部都是平移操作的情况下才成立。如果同时进行其他操作，比如旋转或缩放，那么原点就很重要了。（稍后详述组合变换。）

最后一种变换是扭曲（skew），它稍微复杂一些，然而声明方法并不比你目前见过的难。扭曲一个元素将使其沿一个轴或两个轴变形，如图7-39所示。

```
.box1 {transform: skew(23deg);}
.box2 {transform: skew(13deg,-45deg);}
```

如果只为skew()提供一个值，那么就只有水平（*X*）扭曲，没有垂直（*Y*）扭曲。像平移和缩放一样，也存在skewX()和skewY()值，可以用在希望明确地沿着一个坐标轴进行扭曲的时候（如图7-40所示）。

```
.box1 {transform: skewX(-23deg);}
.box2 {transform: skewY(45deg);}
```

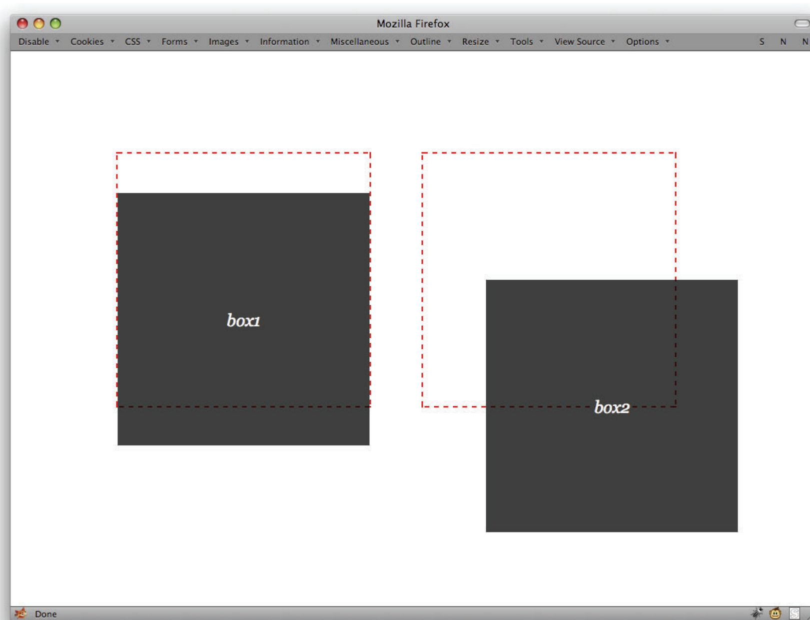


图7-38　平移后的两个不同元素

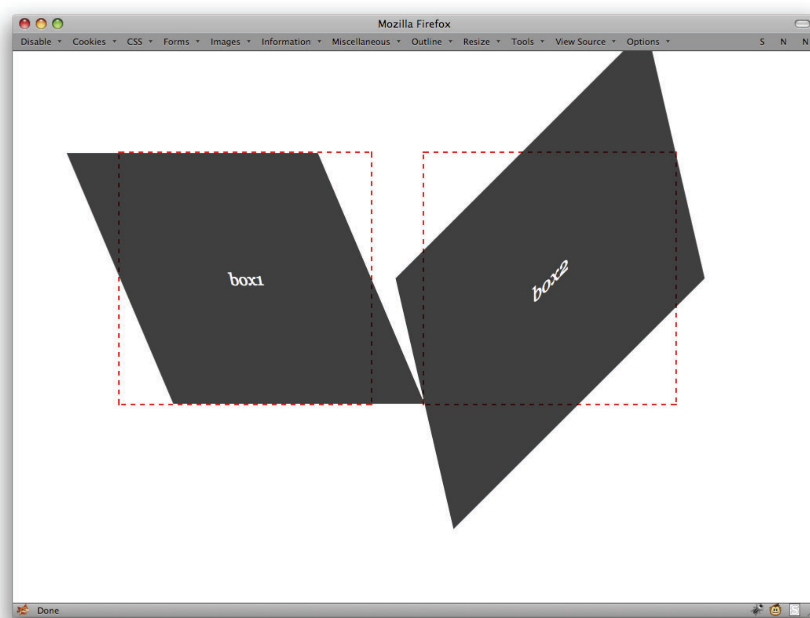


图7-39　两个扭曲的元素

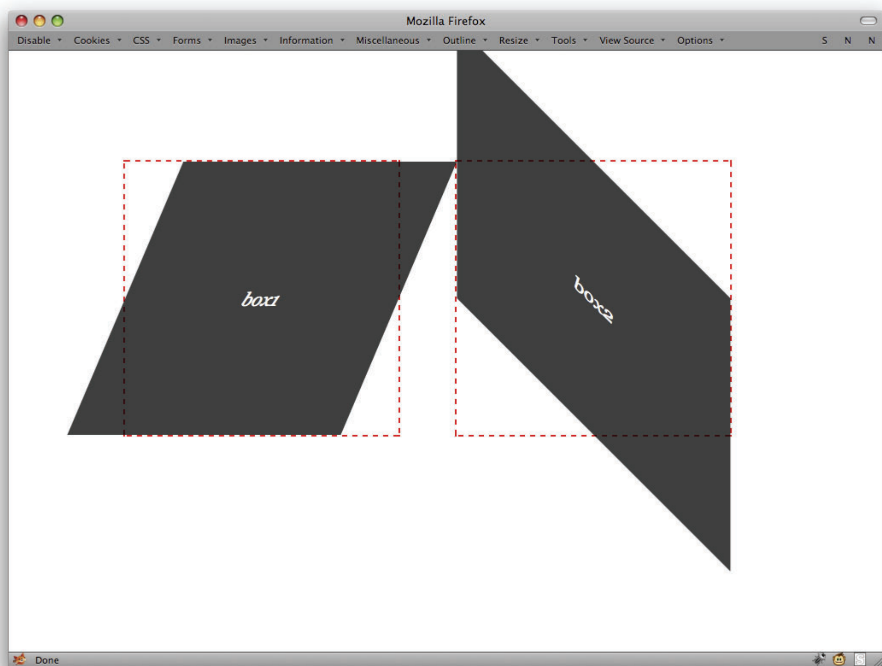


图7-40 两个沿着不同的坐标轴扭曲的元素

下面介绍扭曲的工作原理。假设有两根木杆穿过元素，一个沿着*X*轴，另一沿着*Y*轴。当沿着*X*轴方向扭曲时，*Y*轴会旋转过指定的扭曲角度。是的，在skewX()操作中旋转的是*Y*轴（垂直的）。正的角度是逆时针的，而负角度是顺时针的。这就是前面例子中的第一个框会向右倾斜的缘故：*Y*轴被顺时针倾斜了33.3度。

这个基本情况也会发生在skewY()上：*X*轴会被倾斜特定的角度，正角度时会逆时针倾斜，而负角度时会顺时针倾斜。

这里最有趣的部分就是原点的放置方式。如果原点是在中心，而你为skewX()指定了一个负值的话，那么元素的顶部就会滑向原点的右侧，而底部会滑向原点的左侧。然而，如果把原点改到元素的底部，那么整个元素会从底部开始完全向右倾斜。参见图7-41。

```
.box1 {transform: skewX(-23deg);}
.box2 {transform: skewX(-23deg); transform-origin: bottom center;}
```

垂直扭曲也会产生类似的效果。

那么这些就是你能实现的所有变换类型了。但是如果你想同时使用不止一个的话，该怎么做呢（如图7-42所示）？没问题！只需按照你想让它们发生的顺序把它们列出来。

```
.box1 {transform: translateX(50px) rotate(23deg);}
.box2 {transform: scale(0.75) translate(25px,-2em);}
```

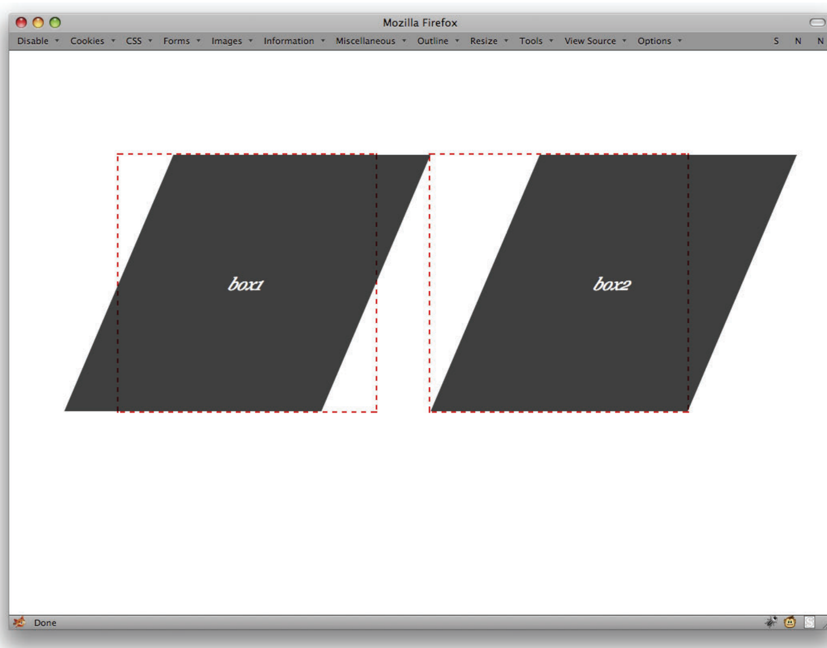


图7-41　两个具有不同原点的元素（已扭曲）

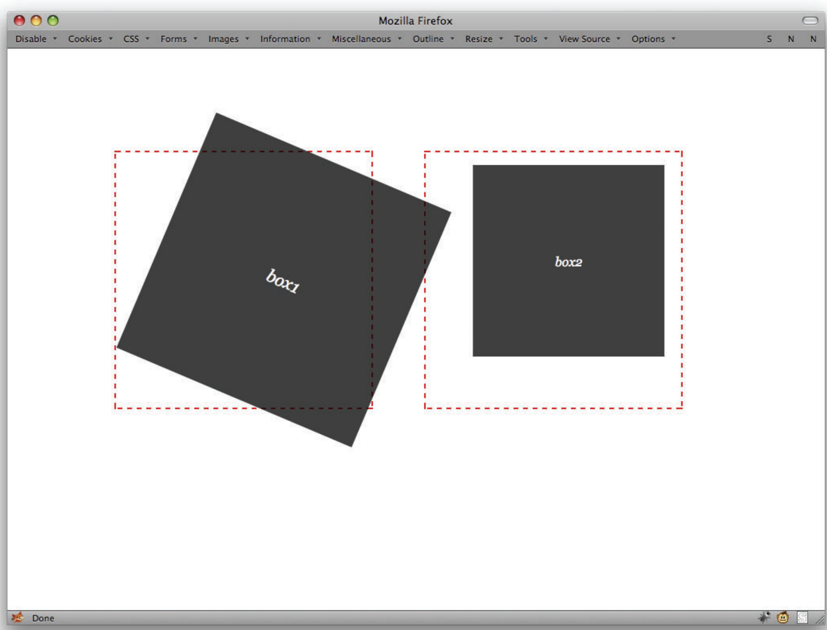


图7-42　同时应用多个变换

在任何情况下，都是从第一个开始，每次执行一个变换。这可以造成非常显著的差异，考虑相同的变换以不同的顺序组合所呈现的效果（如图7-43所示）。

```
.box1 {transform: rotate(45deg) skew(-45deg);}
.box2 {transform: skew(-45deg) rotate(45deg);}
```

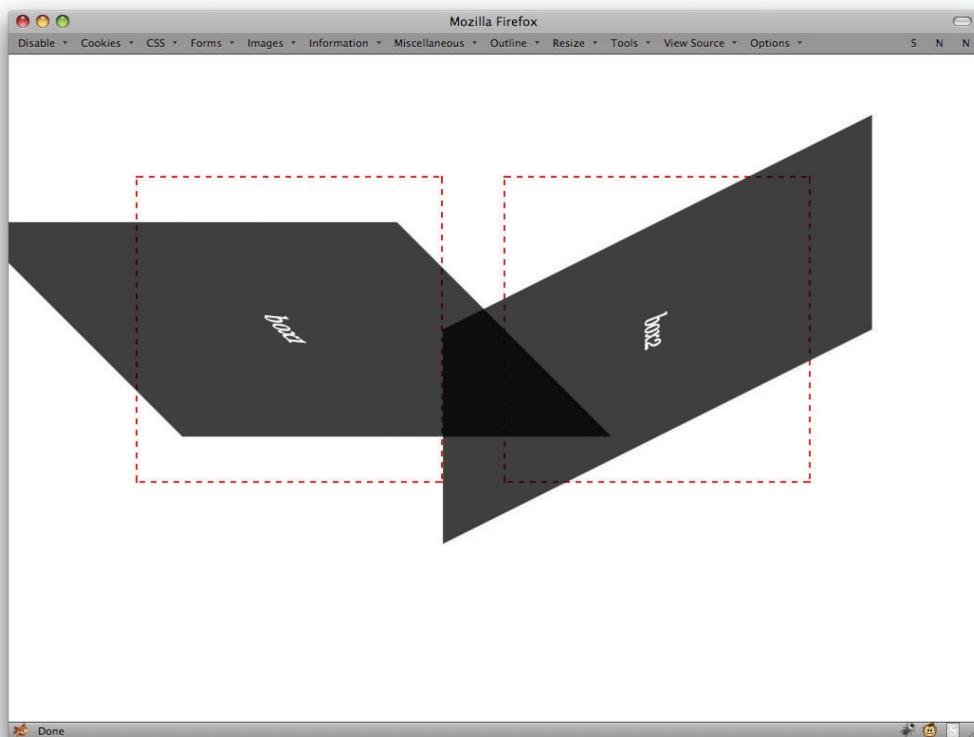


图7-43 由变换值的顺序不同引发的不同效果

这里还有一个需要介绍的变换值类型`matrix()`，这个值类型允许你指定一个包含6部分的变换矩阵，最后两部分定义了平移。下面是一个代码示例，结果如图7-44所示。

```
.box1 {transform: matrix(0.67,0.23,0,1,25px,10px);}
.box2 {transform: matrix(1,0.13,0.42,1,0,-25px);}
```

基本上，前4个数字是简写形式，代表了旋转、扭曲以及缩放一个元素的最终结果，而最后两个数字会平移最终结果。如果理解数学中的矩阵变换，那么你会“爱”上这种方式的。如果不理解，也不要太过担心。回顾一下本节介绍的其他变换类型，通过它们也可以达到同样的效果。

如果你想了解一些关于矩阵变换的知识，这里有两个参考资源：

- http://en.wikipedia.org/wiki/Linear_transformation#Examples_of_linear_transformation_matrices

- http://www.mathamazement.com/Lessons/Pre-Calculus/08_Matrices-and-Determinants/coordinate-transformation-matrices.html

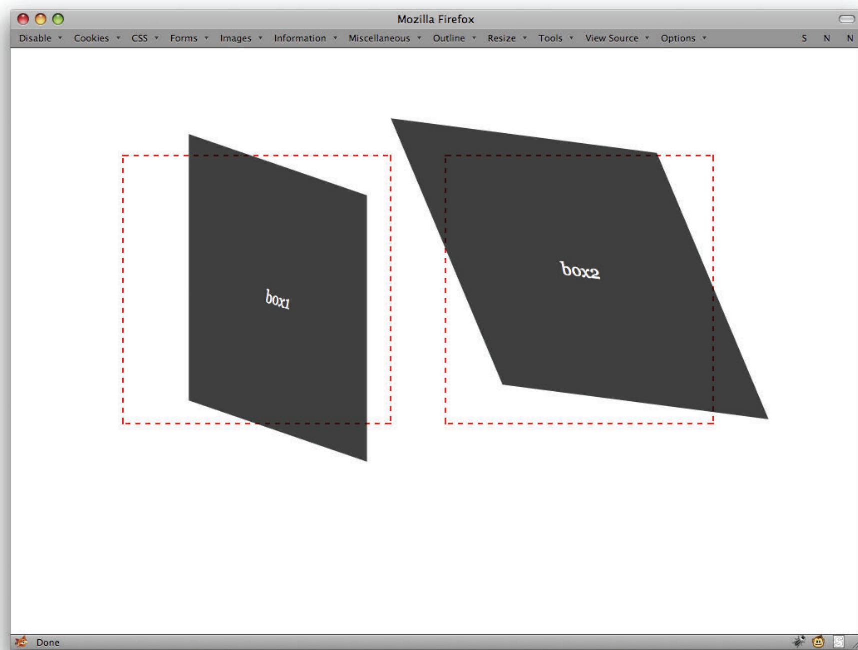


图7-44 矩阵变换

打造现代布局的专业技术

Smashing CSS Professional Techniques for Modern Layout

精彩绝伦的CSS

本书远非只是介绍基础知识，它不仅全面细致地讲解布局与效果，而且展望了HTML5和CSS3的未来。业内很少有人能像Eric A. Meyer一样详细阐明CSS，他在本书中深入分析了普遍适用的实用技术，讲解了如何选用正确的工具、如何通过jQuery使用CSS效果和CSS3技术。

本书主要内容包括：

- 显示或隐藏元素
- 通过XHTML为body或html元素设置背景
- 超过15种布局技巧，包括清除浮动、两栏/三栏布局、伪造栏布局、“唯一布局”、Holy Grail、基于em的布局、流式网格、固定页脚等
- 各种CSS效果，包括CSS弹出框、框冲切、圆角、CSS精灵、滑动门、流式漂白、参差浮动等
- 应用CSS表样式，包括使用表头、主体、脚注、行标题、面向列的样式、表的映射和图形化等
- 使用CSS3元素、多背景支持、RGBA，以及通过jQuery执行CSS3选取操作等

本书适合具有一定的CSS和JavaScript使用经验的Web开发人员学习参考。

Eric A. Meyer　国际公认的HTML、CSS、Web标准方面的专家，自1993年开始从事与Web设计与开发相关的工作。著有《Eric Meyer谈CSS（卷1）》和《Eric Meyer谈CSS（卷2）》。

- CSS专家Eric A. Meyer新作
- 全面解说CSS现代技巧和最佳实践
- Smashing杂志专业策划

WILEY
www.wiley.com

图灵社区：www.ituring.com.cn
新浪微博：@图灵教育　@图灵社区
反馈/投稿/推荐信箱：contact@turingbook.com
热线：(010)51095186转604

分类建议　计算机/网络技术/CSS

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-28479-2
定价：49.00元